НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра автоматики та управління в технічних системах
(повна назва кафедри)

«На правах рукопису»
УДК 004.75

«До захисту допущено»
Завідувач кафедри
_____ Ролік О. І.
(підпис) (ініціали, прізвище)
“___”_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 126 – Інформаційні системи та технології
(код і назва спеціальності)

_____

на тему: Система контролю якості програмного забезпечення

Виконав: студент VI курсу, групи ІА-72мп
(шифр групи)

Соболєв Артем Григорович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н, доцент, Букасов М.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Факультет (інститут) інформатики і обчислювальної техніки
(повна назва)

Кафедра автоматики та управління в технічних системах
(повна назва)

Рівень вищої освіти – другий (магістерський)
(код і назва)

Спеціальність 126 – Інформаційні системи та технології
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О. І. Ролік
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Соболєву Артему Григоровичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Система контролю якості програмного забезпечення»
_____
науковий керівник дисертації Букасов Максим Михайлович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «__» _____ 20___ р. №_____

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження процес контролю якості програмного забезпечення
_____

4. Предмет дослідження математична модель процесу контролю якості програмного забезпечення _____
_____

5. Перелік завдань, які потрібно розробити: сформувати теоретичну базу дослідження; розробити математичну модель процесу контролю якості програмного забезпечення; реалізувати програмне забезпечення для побудови відповідної моделі

6. Орієнтовний перелік ілюстративного матеріалу Діаграма прецедентів програмного забезпечення, Діаграма станів програмного забезпечення, Діаграми діяльності програмного забезпечення, Діаграми послідовності програмного забезпечення, Діаграми кооперацій програмного забезпечення, Діаграма класів програмного забезпечення.

7. Орієнтовний перелік публікацій _____

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Отримання завдання та узгодження вихідних даних	29.10.2018	
2	Вивчення об’єкту дослідження	30.10.2018	
3	Розробка математичної моделі	10.11.2018	
4	Розробка програмної моделі	15.11.2018	
5	Проведення експерименту	25.11.2018	
6	Оформлення документації	02.12.2018	
7	Подання роботи до попереднього захисту	04.12.2018	

Студент _____ А.Г. Соболєв
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ М.М. Букасов
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Дана магістерська робота присвячується розробці системи контролю якості програмного забезпечення за допомогою побудови математичної моделі процесів. Для складання математичної моделі використовувались мережі Петрі, що дало змогу наочно відобразити модельований процес та полегшити розробку моделі.

Магістерська робота виконана на 113 аркушах формату А4, містить 45 рисунків, 46 таблиць, 2 додатки та список використаних джерел з 16 найменувань.

ABSTRACT

This master's thesis is devoted to the development of software quality control system by constructing a mathematical model of processes. Petri's networks were used to compile the mathematical model, which allowed to visualize the simulated process and facilitate the development of the model.

Master's thesis is executed on 113 sheets of A4 format, contains 45 drawings, 46 tables, 2 applications and a list of used sources of 16 titles

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ГІК – Графічний Інтерфейс Користувача

ДСТУ – Державні Стандарти України

ПЗ – Програмне Забезпечення

ПП – Програмний Продукт

ISO – International Organization for Standardization (Міжнародна організація зі стандартизації)

QA – Quality Assurance (Забезпечення Якості)

TQM – Total Quality Management (Загальне Управління Якістю)

ВСТУП

Неухильне підвищення якості випущених програмних продуктів – це основна мета кожного розробника. Питанням якості стурбовані не тільки розробники програмного забезпечення для зарубіжного замовника (рівень вимог до якого завжди був високий), але й організації, що працюють на українського замовника або безпосередньо на український ринок споживачів. Це, в першу чергу, пояснюється загостренням конкуренції розробників, а, крім того, – підвищенням рівня знань споживачів про якість ПЗ і, як наслідок, зростанням їх вимог до якості.

Актуальність роботи полягає в тому, що в останні роки створюються все нові моделі процесів розробки програмного забезпечення. Застосування мереж Петрі в якості математичного апарату моделювання дозволяє враховувати особливість процесів розробки програмного забезпечення – паралелізм виконання дій. Моделювання і аналіз процесів розробки програмного забезпечення не розглядаються при цьому з точки зору тривалості виконання тих або інших дій. Модель аналізується із поведінкової точки зору, розглядається у вигляді послідовності дискретних подій. Відомою галуззю використання мереж Петрі є формальна специфікація та верифікація вимог та дизайну програмних продуктів. Зростає увага до процесів контролю якості програмних продуктів. У цьому напрямі вже створено ряд програм, але вони мають ряд недоліків, більшість з яких можна виявити лише аналізуючи процеси в ході реальної роботи. Один з них – відсутність оцінки часу для аналізу та усунення невідповідностей між реальним процесом та його моделлю. Таким чином, створення математичної моделі процесу контролю якості програм та розробка програмного забезпечення для моделювання та аналізу цього процесу являють собою актуальну проблему як в науковому, так і в прикладному розумінні.

Мета роботи – створення системи, яка буде будувати математичну модель процесу контролю якості програмного забезпечення, яка дозволить знаходити ймовірні помилки та слабкі місця програмних проектів на початкових стадіях їх життєвого циклу.

Об’єктом дослідження є процес контролю якості розробки програмного забезпечення.

Предметом дослідження є математична модель процесу контролю якості програмного забезпечення.

Методи дослідження полягають у тому, що для досягнення поставленої мети в якості методу моделювання процесів будуть використовуватися мережі Петрі. Вони зазвичай використовуються для подання та дослідження процесів на різних рівнях опису моделей. Вибраний метод моделювання відображає специфіку реалізації взаємодії та стан різноманітних ресурсів. Перевага мереж Петрі – чіткий математичний опис моделі, що дозволяє виконувати їх аналіз за допомогою сучасної обчислювальної техніки.

1 ОСНОВНІ ПОЛОЖЕННЯ ПРО ЖИТТЄВИЙ ЦИКЛ, КОНТРОЛЬ ЯКОСТІ ТА ЙОГО МОДЕЛЮВАННЯ

1.1 Основні поняття розробки програмного забезпечення

1.1.1 Життєвий цикл ПЗ

Життєвий цикл програмного забезпечення зазвичай включає часовий інтервал між зародженням ідеї та повним використанням останньої версії.

Неможливо описати загальну структуру життєвого циклу, оскільки розробка та розвиток програмного забезпечення для вирішення різних завдань у різних областях занадто різні. Проте основними поняттями, що описують таку структуру, є діяльність, ролі та артефакти.

Тип активності в життєвому циклі програмного забезпечення – це перелік дій для вирішення завдання або групи завдань, які тісно пов'язані з розробкою та підтримкою програмного забезпечення. Наприклад: аналіз предметної області, дизайн, розробка коду, тестування тощо.

Роль у життєвому циклі програмного забезпечення – це професійна спеціалізація людей, які беруть участь у створенні або підтримці програмного забезпечення, і які поділяють однакові погляди або вирішують ті ж задачі, пов'язані з ним. Наприклад: бізнес-аналітик, архітектор, конструктор інтерфейсів, програміст, менеджер проектів, користувач, адміністратор та інші.

Артефакти життєвого циклу програмного забезпечення – це різні типи інформації, документи та шаблони, створені або використані для розробки та підтримки програмного забезпечення. Артефакти включають: технічні завдання, архітектурний опис, документація користувача тощо. [1, 9]

1.1.2 Якість програмного забезпечення та її характеристики.

Як правило, визначення якості програмного забезпечення використовується у вигляді системи атрибутів (факторів), які можна оцінити за допомогою набору метрик. Такий підхід дозволяє конструктивно оцінити якість програмного забезпечення в цілому і в усіх необхідних аспектах. Фактори та атрибути зовнішньої та внутрішньої якості програмного забезпечення (рисунок 1.1), ISO 9126, описана нижче [6].

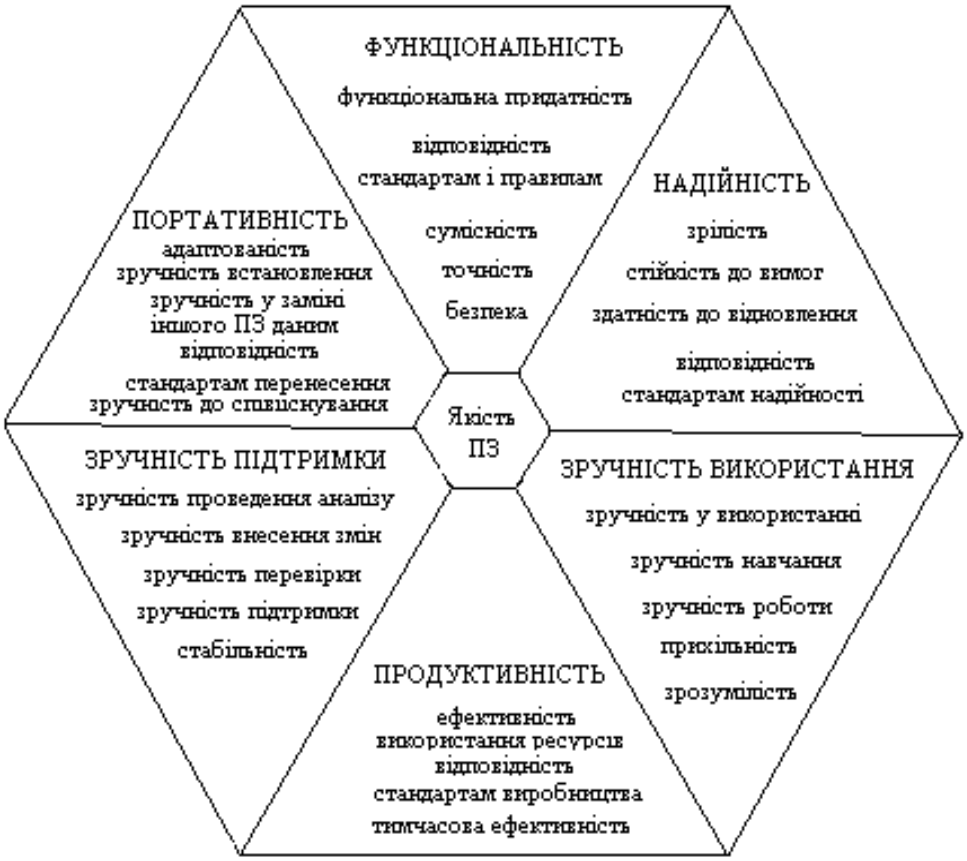


Рисунок 1.1 – Фактори та атрибути зовнішньої та внутрішньої якості програмного забезпечення

Відповідно до цього стандарту, аналіз якості програмного забезпечення враховує три аспекти: думка розробника, яка визначає внутрішню якість програмного забезпечення; думка управління та програмне забезпечення сертифікації для дотримання вимог, що містяться в ньому, в ході якого визначаються зовнішня якість програмного забезпечення та думкою користувачів, які оцінюють якість використання програмного забезпечення. У всіх трьох випадках використовується модель, що складається з цілей або факторів, атрибутів або критеріїв та показників якості.

Цілі (фактори) дозволяють найвищому рівню визначати основні функції, які програмне забезпечення має чи повинно мати. Кожен фактор складається з безлічі атрибутів, які дозволяють більш детально описати бажані або отримані функції. Кожен атрибут підтримується набором показників, які можуть бути використані для кількісного визначення присутності відповідної функції. Рисунок 1.1 показує зовнішню та внутрішню якість запропонованої моделі, яка включає 6 факторів та 27 атрибутів.

Визначення факторів та атрибутів даної моделі розписані нижче [8].

1) Функціональність (functionality) – здатність програмного забезпечення при певних умовах вирішувати завдання, необхідні користувачем. Визначає, що саме робить програмне забезпечення. Нижче розписані її основні складові,

Функціональна придатність (suitability) – здатність вирішення необхідного списку завдань.

Точність (accuracy) – здатність отримати певний та очікуваний результат.

Здатність до взаємодії, сумісність (interoperability) – наявність можливості взаємодіяти з іншими системами.

Відповідність стандартам і правилам (compliance) –програмне забезпечення повинно відповідати стандартам, нормативним та законам або іншим регулюючим нормам в межах держав, де воно розповсюджується.

Безпека (security) – наявність елементів захисту, які не тільки не дозволять розповсюджувати програмне забезпечення нелегальними каналами, а й захистять дані користувачів від зовнішніх або внутрішніх загроз.

2) Надійність (reliability) – здатність програмного забезпечення працювати без помилок, зависань та без перенавантажень систем без необхідності. Нижче розписані її основні складові,

Зрілість (maturity) – це зворотна величина до частоти відмов, зависань, помилок програмного забезпечення.

Стійкість до відмов (fault tolerance) – це здатність програмного забезпечення працювати на заданому рівні у разі відмови або виникнені помилок в процесі роботи.

Здатність до відновлення (recoverability) – це здібність програмного забезпечення повернутися в нормальний стан роботи та підтримувати цілісність даних після відмови або виникненні помилок в процесі роботи.

Відповідність стандартам надійності (reliability compliance).

3) Зручність використання (usability) або практичність – це легкість в освоєнні роботи з програмним забезпеченням та наявність в ньому зручного для користувача інтерфейсу.

Зрозумілість (understandability) – це показник того, наскільки користувач швидко та легко освоїв працю з програмним забезпеченням та освоїв максимально функціонал, що полегшить основну працю з ним.

Зручність навчання (learnability) – це показник того, скільки зусиль користувач затратив на освоєння роботи з програмним забезпеченням.

Зручність роботи (operability) – це показник того, скільки зусиль витрачає користувач в роботі в програмному забезпечені при розв’язані задач в ньому.

Прихильність (attractiveness) – це показник того, наскільки програмне забезпечення може зацікавити та привабити користувача.

Відповідність стандартам зручності у використанні (usability compliance).

4) Продуктивність (efficiency) – це здатність програмного забезпечення працювати в нормальному режимі без використання зайвих обчислювальних або інших ресурсів. Нижче розписані її основні складові,

Тимчасова ефективність (time behaviour) – це здатність програмного забезпечення вирішувати поставлені задачі за певний проміжок часу.

Ефективність використання ресурсів (resource utilisation) – це здатність програмного забезпечення працювати в межах виділених йому на це ресурсів. До них відносяться оперативна пам'ять, мережеві з’єднання, центральний та графічний процесор і т.д.

Відповідність стандартам виробництва (efficiency compliance).

5) Зручність підтримки (maintainability) – це коли в користувача є можливість будь-якої діяльності, що пов’язана з підтримкою програмного забезпечення. Нижче розписані її основні складові,

Зручність проведення аналізу (analyzability) – це коли користувач може провести аналіз помилок або недоліків, а також проаналізувати необхідність змін та наслідки, що вони можуть мати.

Зручність внесення змін (changeability) – це показник того, скільки потрібно витратити(зусиль, часу, тощо) для виконання необхідних змін.

Стабільність (stability) – це показник того, наскільки добре працює програмне забезпечення при внесенні будь-яких змін.

Зручність перевірки (testability) – це показник того, наскільки багато треба затратити ресурси(час, зусилля, тощо) для проведення тестувань або будь-яких перевірок для того, щоб впевнитись, що внесені зміни були без помилок та програмне забезпечення працює без проблем.

Відповідність стандартам зручності підтримки (maintainability compliance).

6) Портативність (portability) – це здатність програмного забезпечення працювати при переносі з одного оточення в інше. До них входять апаратні та програмні аспекти оточення. Нижче розписані її основні складові,

Адаптованість (adaptability) – це здатність програмного забезпечення працювати в різноманітному оточенні(апаратаному або програмному) не виконуючи для цього зайвих операцій.

Зручність встановлення (installability) – це здатність програмного забезпечення бути встановленим чи розгорнутим у деякому оточенні без зайвих операцій та простим методом.

Здатність до співіснування (coexistence) – це здатність програмного забезпечення працювати з іншими програмами, не забираючи в них обчислювальні ресурси.

Зручність у заміні (replaceability) іншого ПЗ даним – це здатність використання програмного забезпечення, яке має той самий функціонал, що дозволяє вирішувати ті самі задачі. Наприклад, Microsoft Office та LibreOffice.

Відповідність стандартам перенесення (portability compliance).

Наведені характеристики описують якість програмного забезпечення з точки зору розробника або керівництва. З точки зору користувача, якість програмного забезпечення, коли воно використовується, повинно відповідати наступним факторам [10]:

1) ефективність (effectiveness) – це здатність вирішувати завдання поставлені користувачем в межах можливостей програмного забезпечення з максимальною точністю;

2) продуктивність (productivity) – це здатність отримати користувачу необхідні результати в межах використаних ресурсів;

3) безпека (safety) – це здатність забезпечити низький рівень ризику загрози життю та здоров’я користувачу або його бізнесу;

4) задоволення користувачів (satisfaction) – це здатність користувачу отримувати задоволення він легкості та зручності праці з програмним забезпеченням.

1.1.3 Контроль якості програмного забезпечення.

Якість програмного забезпечення – це деякі характеристики програмного забезпечення, які потрібні відповідати певним ступеням вимог.

Забезпечення якості – це комплекс заходів, що охоплює всі технологічні етапи розробки, публікації та експлуатації програмного забезпечення, які використовуються на різних етапах життєвого циклу програмного забезпечення для забезпечення якості продукту.

Контроль якості – це набір дій, які виконуються над об’єктом розробки і надають інформацію про стан тестового об'єкта.

Верифікація та валідація – це моніторинг якості програмного забезпечення і виявлення помилок. Маючи одну й ту саму мету вони мають різні правила перевірки властивостей, правил та обмежень, порушення яких вважається помилкою.

Верифікація перевіряє, що певні артефакти, створені під час розробки та обслуговування програмного забезпечення, інші, які раніше були створені або використані в якості вхідних даних, а також відповідність цих артефактів і процесів їх розробки правил і стандартів. Крім того, верифікація перевіряє відповідність між стандартами, специфікаціями, проектними рішеннями, програмним кодом, призначеної для користувача документації і роботою самого програмного забезпечення.

Помилки та неточності, виявлені під час перевірки, можуть бути негайно виправлені.

Валідація перевірятиме узгодженість артефактів, створених і використовуваних при розробці і обслуговуванні програмного забезпечення, а

також потреб користувачів і клієнтів цього програмного забезпечення з урахуванням предметної області.

Це завжди відбувається у співпраці з замовником, користувачем, експертами галузі і іншими зацікавленими особами.

Якість програмного забезпечення – це набір функцій програмного забезпечення, пов'язаних із задоволенням існуючих вимог.

1.1.4 Моделювання процесів розробки програмного забезпечення.

Моделювання процесів є важливою частиною аналізу та вдосконалення програмного забезпечення. Модель процесу – це структурований набір елементів, що описують характеристики ефективного процесу.

Існує кілька способів моделювання процесів, наприклад, з використанням конструкції: діаграми станів, діаграми потоків даних, блок-схеми, а також модель продуктивності процесу та ін.[2]

Діаграма станів використовується для опису стану об'єкта та умов переходу між ними. Опис станів дозволяє точно описувати поведінку об'єкта, коли він отримує різні повідомлення та взаємодіє з іншими. Діаграма станів використовується для демонстрації динамічної моделі елементів та фокусується на специфічних аспектах взаємодії. Як правило, діаграма стану використовується для моделювання поведінки активних класів.

Діаграма потоків даних являє собою графічний інструмент для відображення потоку інформації і перетворень, яким дані піддаються, як вони проходять від вхідного отвору до вихідного отвору системи. Діаграма може бути використана для представлення програмного продукту на будь-якому рівні абстракції.

Блок-схеми – це універсальна форма представлення або запису алгоритмів. Універсальність пояснюється тим, що вона не визначає пояснюваних абстракцій в блоках, навіть за допомогою звичайних слів. Блоки

являють собою лише шаблон для опису дій, які вирішуються, і зв'язок між блоками визначає послідовність цих дій.

Модель продуктивності процесу виконує аналіз ефективності процесів та продуктів. Він систематизує отриману інформацію з різних функцій та вимірює параметри виробництва. Ці результати можуть бути представлені у формі звіту або представлені у формі постійної оцінки показників ефективності.

Вибір типу моделі залежить від цілей процесу покращення завдань, які потрібно вирішити, та визначення аспекту модельованого процесу. Наприклад, потрібно проаналізувати послідовність дій, ресурсів, даних або функцій, що виконуються при моделюванні.

1.2 Мережі Петрі, як спосіб математичного моделювання

Мережі Петрі являє собою математичну абстракцію для представлення дискретних розподілених систем.

Мережі Петрі були вперше запропоновані в докторській дисертації Карла Петрі в 1962 році, а потім вони були розвинуті в роботах таких науковців, як Тадау Мурат, Курт Дженсен, Віталій Котов, Анатолій Сліпцов. Навіть було організовано щорічну конференцію «Застосування і теорія мереж Петрі», публікується бюлетень «Новини мереж Петрі» у Бонні, реалізовано кілька сотень моделювальних систем для різних програмних і апаратних платформ, а також мережеві процесори Петрі.

Мережа Петрі описується сукупністю об’єктів [3]:

$$PN = (P, T, F, W, M_0),$$

де $P = \{P_1, P_2, ..., P_m\}$ – скінченна множина позицій; $T = \{t_1, t_2, ..., t_n\}$ – скінченна множина переходів; $F \subseteq P \times T \cup T \times P$ – множина орієнтованих дуг, які зв’язують позиції та переходи, що є вершинами графа; M_0 – початкове маркування, яка забезпечує наявність умов для запуску переходів.

Графічно мережа Петрі інтерпретується як спрямований мультиграф. Мережа Петрі – це мультиграф, оскільки допускає існування кратних дуг з однієї вершини до іншої. Оскільки дуги спрямовані, це підкреслює мультимографію. Оскільки дуги можуть тільки з'єднувати вершини різних типів, такий граф є дводольним орієнтованим мультиграфом. При зображені мережі Петрі графічним способом переходи зображуються квадратами або планками (*t*), а позиції відповідно кружками (*P*).

Позиції можуть містити теги, які можуть переміщатися по мережі. Розташування тегів зазначено маркуванням. Переміщення тегів здійснюється по орієнтованим дугам за допомогою переходів при запуску. Кількість міток, видалених або доданих в позицію, визначається вагою дуги. Робота мереж Петрі полягає в тому, щоб переміщати теги з однієї позиції в іншу. Мітки розташовані в кружках і контролюють виконання мережевих переходів. Перехід починається з видалення міток з його вхідних позицій і створення нових міток в його вихідних позиціях [4, 12].

В даний час мережа Петрі використовується в основному для моделювання. Моделювання в мережах Петрі виконується на рівні подій. Визначаються, які дії відбуваються в системі, які стани передують цим діям, і в які стани система перейде після цих дій. Таким чином, модель мережі Петрі використовується для відображення і аналізу причинних зв'язків в системі. Причини (умови) являють собою позиціями, які можуть мати теги, а події являють собою переходами, які можуть мати пріоритет.

Моделювання процесів розробки програмного забезпечення з використанням мереж Петрі дає наступні результати [5, 11]:

1. визначити помилки в логіці вихідної моделі процесу, які можуть привести до одночасного запуску однієї і тієї ж операції, що призведе до збільшення вартості ресурсів, збільшення часу проектування і порушення логіки моделі процесу;

2. переконатися, що всі робочі продукти в моделі процесу створені тільки один раз і присутні в кількості одного екземпляра на протязі виконання тієї чи іншої дії. Це забезпечить відсутність дублювання інформації про проект і, при необхідності, усуне додаткові витрати, необхідні для цього;

3. визначити можливі тупики та усунути їх. Тупик в мережі Петрі являє собою переходом (або набором переходів), який не може бути запущений. Виконання моделі процесу не повинно бути заблоковано без можливості розширення. Модель процесу не повинна містити блокуючих ситуацій;

4. оптимізувати модель процесу розробки програмного забезпечення. Оптимізація здійснюється поетапно шляхом вирішення наступних підзадач: підвищення рівня паралелізму, пошуку пасивних позицій і переходів та їх виключення;

5. важливою якісною характеристикою моделі процесу є рівномірний розподіл навантаження ресурсів. Аналіз цієї функцію дозволить визначити простої ресурсів, які можливі при реалізації проекту, щоб визначити мінімальні, максимальні, середні та інші кількісні характеристики їх завантаження.

Перевага мереж Петрі полягає в чіткому математичному описі моделі. Це дозволяє їх аналізувати з використанням сучасних комп'ютерних технологій.

1.3 Постановка задачі

Контроль якості програмного забезпечення є невід'ємною частиною виробничого, складного технічного і організаційного процесу. Основним завданням є забезпечення виробництва, що відповідає потребам споживачів.

Багато років тому назад питання контролю якості програмного забезпечення було майже не актуальним, оскільки вартість програмного забезпечення була незначною. Однак, поступово програмне забезпечення ставало більш складним і дорогим. Сьогодні вартість створення, розробки і підтримки зросла в десятки разів і тим більше процес розробки стає педантичнішим. Складність програмного забезпечення призвела до збільшення кількості програмних помилок, які вплинули на збиток, викликаний цими помилками. Через ці складнощі було необхідно шукати і виправляти ці помилки на різних етапах розробки програмного забезпечення, скорочувати час їх виправлення і скорочувати час до завершення проекту. Слабкі сторони і можливість їх виникнення повинні постійно виявлятися.

Для вирішення цих проблем постійно розробляється нове програмне забезпечення, але всі вони орієнтовані на великі підприємства і економічно затратні. Наш програмний продукт має простий графічний інтерфейс користувача і призначений для простих програмних проектів.

За допомогою моделювання процесу контролю якості розробки програмного забезпечення, яке ми реалізуємо за допомогою мереж Петрі, можна виконати аналіз математичної моделі за допомогою сучасних комп'ютерних технологій. На основі створеної математичної моделі буде створено програмне забезпечення, яке дозволить шукати помилки в програмах під час розробки та визначати час, необхідний для верифікації цього програмного забезпечення.

1.4 Висновки

Такі поняття, як контроль якості, верифікація та валідація, тестування дуже близькі і відіграють важливу роль в життєвому циклі програмного забезпечення. Вони впливають на час, що минув від початку ідеї до її повної реалізації.

Насамперед, поняття контролю якості – це серія дій, що виконуються на випробуваному об'єкті, для отримання інформації про стан цього об'єкта під час розробки. Для контролю якості ми використовуємо мережі Петрі як графічний та математичний інструмент моделювання систем.

Мережі Петрі є дуже потужним і широко використовуваним інструментом для вивчення різних систем. Теорія мереж Петрі дозволяє моделювати систему графічно. За допомогою цієї інформації можна оцінити та вдосконалити систему або змінити її.

2 РОЗРОБКА МАТЕМАТИЧНОЇ МОДЕЛІ ПРОЦЕСУ КОНТРОЛЮ ЯКОСТІ ПРОГРАМ ЗА ДОПОМОГОЮ МЕРЕЖ ПЕТРІ

2.1 Аналіз процесу контролю якості програмного забезпечення

Моделювання процесів контролю якості повинно допомагати з аналізом актуальності поточних та запланованих дій.

Якість не абсолютна, бо це лише суб'єктивна оцінка кожного користувача або розробника. Процес контролю якості дозволяє виявляти помилки своєчасно, але не може гарантувати повну точність та коректну працю програмного забезпечення. Загалом термін "якість" обмежується такими поняттями, як точність, надійність та безпека. Процес контролю якості програмного забезпечення – це комплексний пошук і творчий процес для синтезу структури системи.

Завдання синтезу структур включає: синтез структури керованої системи, а саме визначення оптимального складу та сполучення його елементів, оптимальне розділення об'єктів на конкретні підмножини з заданими властивостями; синтез структури керуючої системи використовується для вибору кількості рівнів і підсистем (ієрархії управління), методів узгодження завдань підсистем різного рівня, що забезпечує прийняття рішень; оптимальне розподіл функцій між рівнями та вузлами системи; вибір структури систем передачі та обробки даних.

2.1.1 Мережа Петрі, як інструмент контролю якості програмного забезпечення

Мережі Петрі – це потужний і широко використовуваний інструмент дослідження для різних систем.

Моделювання мереж Петрі дає змогу описувати елементи системи, їх взаємодію, а також реакцію всієї структури та її компонентів на вхідні дії. В

широкому значені моделювання відноситься до процесу створення логіко-математичної моделі досліджуваної системи, яка описує її структуру та поведінку, і зазвичай приймає форму комп'ютерної програми, а також експерименти з комп'ютерною моделлю для отримання даних про роботу системи протягом певних проміжків часу[12].

При створені моделі за допомогою мережі Петрі завдання полягає у тому, щоб створити симуляційну модель, яка повністю описує поведінку системи або процесу, що моделюється. Через модель, отриману за допомогою мережі Петрі, візуально важко виявити помилки та неточності її побудови. Отримана модель повинна бути ретельно проаналізована для її правильної та адекватної роботи. В іншому випадку помилки моделювання не будуть виявлені та виправлені [3].

Методи аналізу мереж Петрі є комплементарними, їх синтез забезпечує інформацію про точність побудови моделі. Щоб забезпечити найбільш повний зв'язок між моделлю мережі Петрі та функціонуванням реальної системи або процесу, модель, створена таким чином, щоб вона описувала час, необхідний для реалізації конкретних подій та процесів. Відсутність записів про затримки в роботі мережі Петрі не дозволить адекватно оцінити модель.

Мережі Перті є однією з найбільш практичних і популярних мов моделювання, оскільки це формалізована мова, яка може моделювати будь-які процеси(паралельні, тимчасові, тощо) [6].

Інтерпретація мереж Петрі базується на поняттях умов і подій. Стан системи описується низкою подій. Для того, щоб подія відбулася, необхідно , щоб виконувалися певні умови (передумови). Поява подій може призвести до реалізації умов (постумов). В мережі Петрі, умови моделюються позиціями *P*, події являють собою *T*-переходами. Передумови представлені вхідними позиціями відповідних переходів, а постумови вихідними позиціями.

Так, як події, що моделюються мережею Петрі миттєві та не відбуваються одночасно, та їх взаємозв'язок асинхронний, це є практичним

інструментом для моделювання безлічі взаємопов'язаних і паралельних процесів. [7]

2.1.2 Мета моделювання процесу контролю якості

Моделювання процесу контролю якості розробки програмного забезпечення і послуг є важливою частиною їх аналізу і дозволяє виявляти дефекти і недоліки на ранніх етапах життєвого циклу проекту, зводячи до мінімуму час, що витрачається на їх виправлення. Для вирішення цієї проблеми використовується формалізм мереж Петрі, який дозволяє аналізувати математичну модель з використанням сучасних комп'ютерних технологій.

На основі створеної математичної моделі створюється програмне забезпечення, що дозволяє знаходити дефекти програм на етапах розробки і визначати необхідний час для перевірки цього програмного забезпечення.

Сьогодні для участі в процесі контролю якості було створено кілька програмних пакетів, заснованих на теорії мереж Петрі, але всі вони призначені для конкретних завдань в конкретних компаніях, обмежуючи доступ користувачів до них. Планується створення універсального програмного забезпечення для контролю якості продукції. Це було зроблено шляхом аналізу якості програмного забезпечення, вивчення загальної інформації про мережі Петрі[2-3] та поліпшення математичної моделі процесу контролю якості[1, 2] без урахування необхідного часу, що дуже важливо для планування строків розробки програмного забезпечення.

2.2 Моделювання процесу аудиту якості

2.2.1 Аудит якості: призначення і види

Аудит якості – це систематичний та незалежний аналіз, який допомагає визначити релевантність діяльності та результатів з точки зору якості запланованих заходів, ефективності заходів впровадження та їхню здатність досягти поставленої мети.

Аудит якості поділяється на внутрішній та зовнішній. Внутрішній аудит проводиться відповідно до внутрішніх вимог організації. Аудит проводиться співробітниками або аудиторами, які не є працівниками цієї організації.

Основним фактом внутрішнього аудиту є те, що аудитори тут з'являються як незалежні організації. Бажано, щоб вони спілкувалися з персоналом відділу, в якому проводять аудит. Внутрішнє забезпечення якості полягає не стільки в визначенні невідповідностей, скільки у визначенні їх причин, а також оцінку необхідності та можливості запобігання та коригування дій. Це суттєва різниця між аудиторською діяльністю та моніторинговими або контрольними діями, що виконуються для виявлення суперечностей.

Зовнішні аудити виконуються для задоволення вимог організації щодо зовнішньої діяльності. Цей огляд проводиться незалежними експертами, клієнтами та іншими, щоб продемонструвати, що система якості відповідає певним вимогам. Випробування можуть проводитися до укладання договору або до видачі ліцензії на певний вид діяльності, тощо.

Внутрішній аудит системи якості може мати такі причини:

- плановий аналіз ефективності систем якості;
- потреба удосконалити систему якості;
- оцінити заходи, що були проведені з метою поліпшення якості;
- необхідність визначити слабкі місця системи, що можуть створити

проблеми в питанні якості продукції.

Аналіз ефективності системи якості проводиться в плановому порядку, як правило з частотою один раз на рік. Аудит стосується всієї системи якості. Результати цього аудиту є основою для офіційної оцінки вищим керівництвом ефективності системи якості, відповідності її політиці і цілям в області якості. Відповідно до результатів аудиту та офіційної оцінки системи управління якістю, буде розроблена програма щодо її підвищення та створена система коригувальних та запобіжних дій.

Заходи щодо поліпшення системи якості здійснюються на регулярній основі, що відповідає принципу постійного поліпшення якості загальної концепції TQM. Аудит може проводитися частіше, ніж щорічний аналіз якості системи якості, і не охоплювати всю систему, а її лише окремі компоненти.

Для внутрішнього аудиту керівником перевірки є менеджер по якості. Під час підготовки аудиту керівництво компанії і особа, відповідальна за план аудиту, перевіряють процеси, організаційні одиниці і елементи системи якості. Тема і обсяг аудиту визначаються керівництвом відповідно до інформаційних вимог.

Необхідність зовнішньої перевірки якості може існувати з наступних причин:

1. підтвердження відповідності вимогам законодавства;
2. за вимогою замовника;
3. за вимогою органів, що видали сертифікат якості.

Діяльність, пов'язана з певними вимогами до надійності продукції, регулюється законом. У цьому випадку влада має компетенцію ініціювати аудити і вимагати надати акт аудиторської перевірки, що були виконані незалежними експертами.

Зовнішній аудит означає, що аудит якості виконується клієнтом продукту або організацією, якій він довіряє. Ініціатива може виходити з обох

сторін. Як правило, договір визначає, яка організація буде перевірятися, коли і за чий рахунок.

Аудиторські перевірки є свідченням ефективності системи якості та проводяться органом по сертифікації при видачі, поновлення або анулювання сертифіката якості.

Аудит якості в основному застосовується до системи якості. У той же час об'єктами аудиту можуть бути елементи системи якості, такі як процеси або продукти. Іспит може бути згрупований відповідно до мети іспиту в такий спосіб:

- аудит якості системи;
- аудит якості продукту;
- аудит якості процесу, методу.

Перевірка якості кінцевого продукту виконується, як показано на рисунку 2.1, після того, як продукт був протестований і готовий до впровадження. Завдання такого аудиту – оцінити продукт з точки зору клієнта. Метою підсумкового аудиту продукту є виявлення невідповідності в процесі розробки цього програмного контролю та процесу контролю якості програмного забезпечення.

Рисунок 2.1 – Аудит якості готового продукту

Перевірці підлягають наступні елементи під час аудиту якості готового продукту:

- характеристики якості продукту;

- якість роботи програмного забезпечення;
- відповідність документації, що супроводжує продукт, установленим вимогам.

Аудит якості продукту в процесі розробки здійснюється на регулярній основі в найважливіших ділянках робіт або після виникнення питань з приводу якості, доки не будуть виявлені причини невідповідностей. Проблеми з якістю можна виявити в попередніх перевірках. За типом та категорією помилок можна визначити ділянки з невідповідністю. Проведені цільові аудити якості продукції в цих місцях дозволяють визначити причини відхилень та оцінювати ефективність прийнятих заходів.

2.3 Побудова моделі контролю якості програмного забезпечення

Створимо модель планування та якості аудиту, яка покращує цей процес, виявляючи неефективні елементи, помилки та інші проблеми. Процес моделювання є важливою частиною їх аналізу. Модель важлива, оскільки вона є вихідною точкою для загального аналізу та бачення, а також використання досвіду всіх учасників для створення моделі процесу.

Аудит якості є об'єктивною оцінкою відповідності процесів, продуктів та послуг. Це є частиною процесу забезпечення якості (QA), який включає:

- визначення цілей якості та плану управління в організації;
- відстеження та збір метричної інформації та даних проекту;
- проведення аудитів якості в проектах.

Об'єктивність оцінки відповідності забезпечується за допомогою використання критеріїв, які продовжують перевіряти показники ефективності шляхом інституціоналізації виробничої практики. Використання критеріїв означає, що тестові листи містять списки елементів, які потрапляють під перевірку, прийняті всіма учасниками тесту.

Почнемо зі створення простої моделі цієї процедури як одного з видів діаграм діяльності [7]. Необхідні визначення такої моделі:

- процес (Process) – це певний набір активностей, шо мають вхідні та вихідні дані, виконуються послідовно. Наприклад: аналіз вимог, проектування архітектури програми, системне тестування, проведення контролю якості і т.п;
- активність (Activity) має чітке уявлення про завдання і, в цілому, атомарному природу, яка знаходиться в зоні відповідальності однієї людини або групи. Наприклад: планування аудиту, проведення аудиту, реєстрація результатів, усунення невідповідностей, виявлених під час аудиту якості;
- роль (Role) – це певна зона відповідальності.
  Наприклад: менеджер проекту, тестовий інженер, інженер по забезпеченню якості;
- умова (Condition) – 1) вхідна умова, що виконується перед початком процесу або дії; 2) вихідна умова, що виконується по закінченню процесу;
- поставка (Deliver) – результат процесу або діяльності;
- виняток (Exception) – зміна процесу або дії, результатом якої є очікувана або навпаки неочікувана подія. Наприклад: узгоджене відхилення від стандартного процесу, процедура тимчасової зміни виконавця певної ролі; узгоджене відхилення від стандартного процесу, процедура тимчасового заміщення виконавця певної ролі;
- взаємодія (Communication) поділяється на два види формальна чи неформальна. Формальна взаємодія, наприклад, це процедура затвердження постановки проектним менеджером.

  Неформальна взаємодія, наприклад, це обмін електронною поштою з приводу двозначності документу.

На підставі вищезгаданих визначень можна створити модель процесу перевірки якості (рис. 2.2). Є елемент моделі "Процес" (на рисунку він зображений овалом), – це передумова планування аудиту якості, а передумовою є усунення всіх виявлених аномалій. На початку процесу контрольний список узгоджений і налаштовується для поточного проекту. Результатом є звіт про перевірку та виправлення виявлених помилок[13, 14].

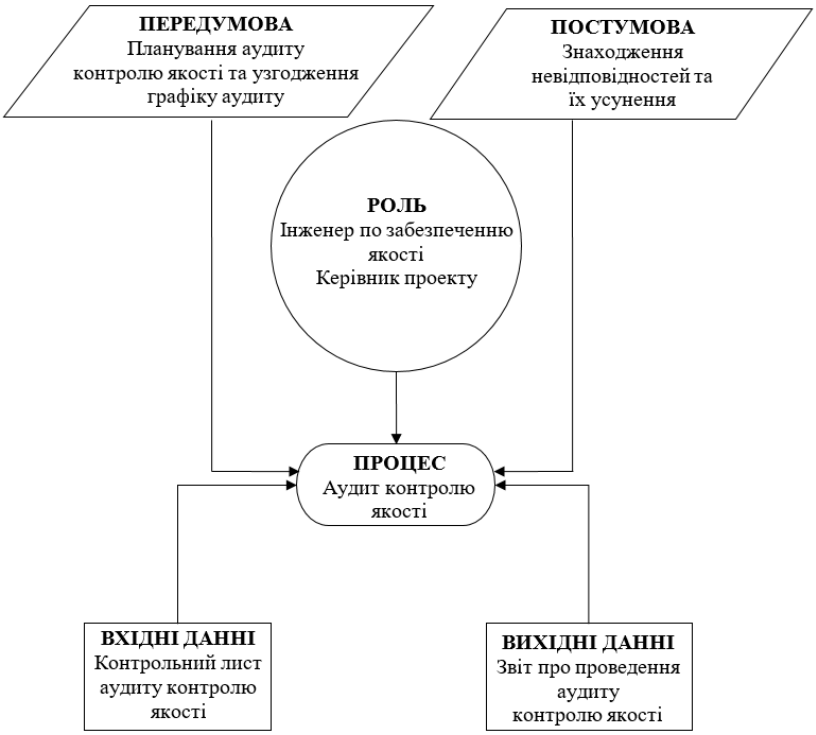


Рисунок 2.2 – Модель процесу аудиту контролю якості

Процес розробки програмного забезпечення і теорія мереж Петрі не схожі один на одного. Індустрія програмного забезпечення шукає рішення для пошуку рішень, в той час як мережа Петрі вимагає досліджень для вивчення програм і властивостей конкретної математичної моделі. Однак системи моделювання та аналізу мереж Петрі можуть використовуватися на різних етапах процесу розробки програмного забезпечення. [15]

Прикладом є використання мереж Петрі для моделювання процесу контролю якості для аналізу, поліпшення і налаштування. Аналіз

моделювання мережі Петрі може допомогти виявити помилки на ранній стадії розробки програмного забезпечення. Ця фаза дуже важлива, тому що якість продукту залежить від якості виробничого процесу.

2.3.1 Моделювання процесу контролю якості за допомогою мережі Петрі

На рисунку 2.4 показана модель процесу перевірки якості. Ролі процедури аудиту: інженер із забезпечення якості, менеджер проекту. Рисунок розділено навпіл, у верхній частині зображені дії інженера по забезпеченю якості, а в нижній керівника проекту.

Ініціалізація якості аудиту передбачає забезпечення передумов для проведення та введення даних таких, як план аудити проекту і попередньо узгоджених контрольних листів аудиту. Кожен контрольний лист адаптований до проекту та схвалений менеджером. Впровадження аудиту та його контрольний перелік будуть здійснюватися з урахуванням специфіки проекту та проведення аудиту більш цілеспрямовано [6].

Нижче приведені кроки процедури, що представлена на рисунку 2.3:

1) інженер із забезпечення якості повинен встановити графік проведення перевірки якості з урахуванням всіх правил і вимог;
2) менеджер проекту аналізує узгоджений розклад і приймає його;
3) проведення аудиту контролю якості;
4) дані моніторингу зберігаються в базі даних;
5) аудиторський звіт створюється і відправляється керівнику проекту для затвердження;
6) визначаються невідповідності, виявлені в ході аудиту;
7) аналіз невідповідностей і часу на їх виправлення;
8) виправлені знайдених помилок і невідповідності;
9) інженер із забезпечення якості перевіряє виправлені невідповідності.

10) після усунення всіх невідповідностей менеджер підтверджує завершення аудиту якості;

А вже на наступному етапі модель цього методу представлена з використання формалізму звичайних мереж Петрі з однокольоровими фішками, рисунок 2.4.

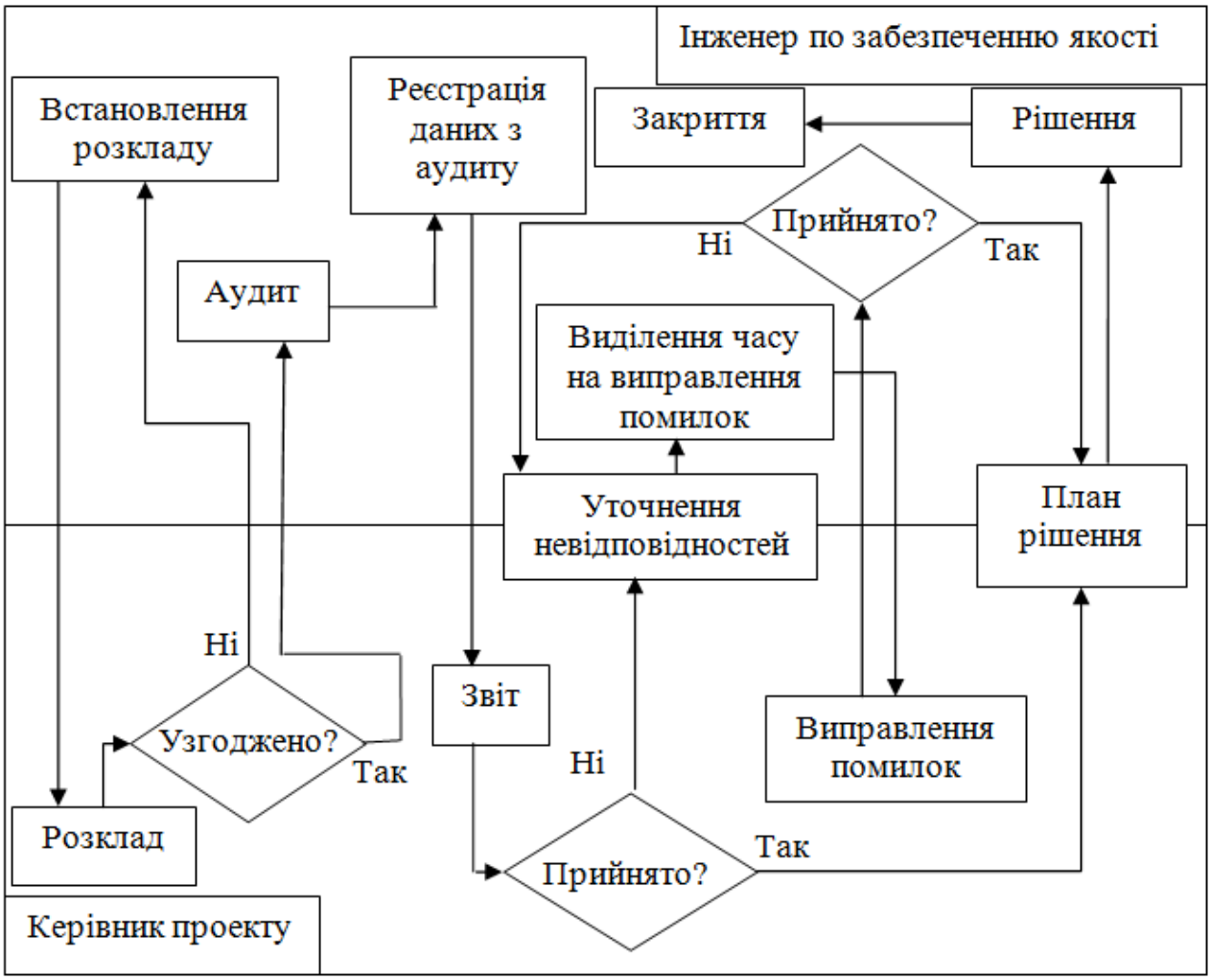


Рисунок 2.3 – Схема проведення аудиту якості

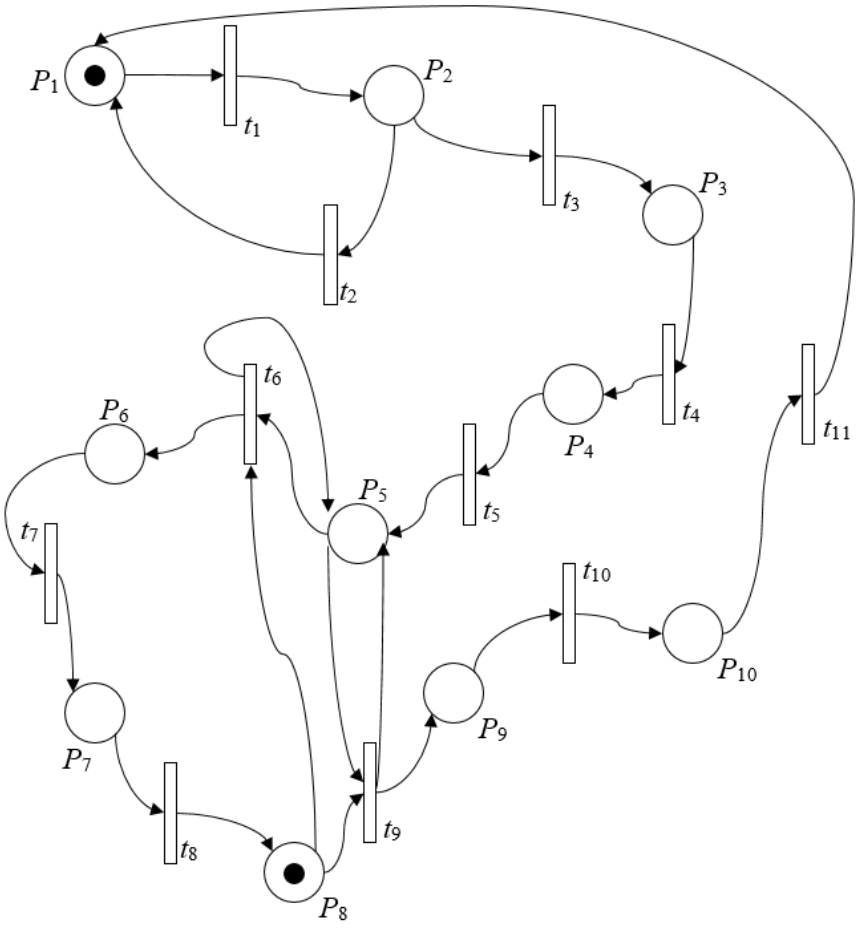


Рисунок 2.4 – Удосконалена мережа Петрі, що моделює проведення аудиту якості

Інтерпретація позицій мережі Петрі приводиться в таблиці 2.1.

Позиції представлені, як умови (*P*), в яких виконується дія, тобто перехід (*T*). Якщо перевірка якості пройшла успішно, всі помилки були виправлені і затверджені керівництвом, схема стає оригінальною. Тобто фішки в позиціях повертаються в початкове положення, циклів немає, і процес може початися спочатку.

Інтерпретація переходів мереж Петрі дана в таблиці 2.2. Перехід спрацьовує, коли відповідна умова виконується. У деяких випадках, залежно від виконання умов, може працювати один із декілька варіантів переходів.

Таблиця 2.1 – Інтерпретація позицій мережі Петрі

Позиції мережі	Призначення фішок в маркірованій мережі	Початкове значення
P_1	Умова початку аудиту контролю якості	1
P_2	Умова узгодженості розкладу проведення аудиту	0
P_3	Умова для проведення аудиту та реєстрації даних з аудиту у вигляді звіту	0
P_4	Умова подання інженером звіту керівникові проекту	0
P_5	Перевірка виконання всіх необхідних умов для складання плану рішень	0
P_6	Умова виділення часу на виправлення помилок та невідповідностей	0
P_7	Умова виправлення помилок та невідповідностей, виявлених в ході аудиту	0
P_8	Умова узгодження виправлених невідповідностей	1
P_9	Умова прийняття рішення по результатам аудиту	0
P_{10}	Умова завершення аудиту	0

Таблиця 2.2 – Інтерпретація переходів мережі Петрі

Переходи мережі	Зміст подій запуск-спрацювання переходу
t_1	Складання розкладу для проведення аудиту
t_2	Подія відхилення наданого розкладу
t_3	Подія надання дозволу для проведення аудиту контролю якості
t_4	Проведення аудиту та оформлення звіту
t_5	Подія узгодження звіту з керівником проекту
t_6	Уточнення невідповідностей, що були виявлені під час проведення аудиту контролю якості

Продовження таблиці 2.2

t_7	Виділення часу для виправлення помилок, зроблених під час проведення аудиту
t_8	Виправлення помилок та невідповідностей
t_9	Складання плану рішень
t_{10}	Прийняття рішення по результатам аудиту
t_{11}	Подія завершення аудиту якості

2.4 Висновки

Розроблено математичну модель процесу контролю якості за допомогою мережам Петрі. Модель важлива, оскільки вона є вхідною точкою для аналізу, загальної мови та бачення, а також для використання досвіду кожного, хто бере участь у створенні моделі процесу. Крім того, ця модель допомагає виявляти помилки та недоліки на різних етапах життєвого циклу проекту, тим самим мінімізуючи витрати на коригування. Модель процесу описує аналіз пробілів, виявлений під час проведення аудиту, і погоджується з інженером із якості.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОНТРОЛЮ ЯКОСТІ

3.1 Вимоги до програмного забезпечення

3.1.1 Функціональни вимоги

Функціонування ПЗ не повинно призводити до збою системи та порушень роботи в системі.

1) Функція, яка надасть змогу побудувати модель у вигляді мережі Петрі. Вона реалізовується графічно, для простоти та зручності у користуванні. Необхідно надати ряд елементів, з яких може складатися мережа та інструкцію по кожному з них для звичайного користувача. В цьому режимі повинна спрацьовувати функція рисування на головному екрані обраних елементів в тих місцях, в яких зазначив користувач (визначаються координати точки в якому потім рисується необхідний елемент мережі). Коли елементи проставлені їх необхідно з'єднати напрямленими лініями зв'язку, що можуть з'єднувати лише елементи різних типів: позиції з переходами, або навпаки. Таким чином ми отримуємо схематичне представлення умов та дій модельованої системи у вигляді мережі Петрі. Для завершення побудови мережі потрібно розставити маркування в необхідні позиції та поставити пріоритети переходів в тих випадках, коли при виконанні однієї умови може виникати декілька подій. Будування мережі завершено, можна переходити до перевірки її працездатності.
2) Функція перевірки працездатності мережі, яка надасть змогу дізнатись чи правильно складена модель в цілому. В цьому режимі програма повинна виявити помилки в логіці вихідної моделі процесу, які можуть вести до багаторазової ініціації однієї і тієї ж діяльності в один момент часу, що, у свою чергу, веде до збільшення витрат ресурсів, збільшення проектного часу і порушення логіки моделі процесу. Також в цьому режимі повинні виявлятись

можливі тупикові ситуації, які необхідно усунути. Тупик в мережі Петрі – це перехід (або множина переходів), який не може бути запущений. Виконання моделі процесу не має блокуватися без можливості подальшого продовження. Модель процесу повинна бути вільна від наявності тупикових ситуацій.

3) Функція матричного аналізу, яка надасть змогу перевірити досяжність маркування поточною мережею. Аналіз проводиться матричним методом і в результаті ми отримаємо перелік маркувань, необхідних для досяжності потрібного маркування. Якщо ж маркування не досяжне то отримаємо повідомлення про помилку.
4) Функція побудови дерева покриваючих маркувань, яка надасть змогу побудувати схематичне дерево. На основі дерева проводиться аналіз мережі, який необхідний для виявлення помилок в мережі.

3.1.2 Нефункціональні вимоги

Нижче наведено перелік нефункціональних вимог до ПЗ.

1) Можливість створення нового документу (файлу), в якому можна буде побудувати модель для аналізу використовуючи мережі Петрі.
2) Можливість використовувати програмне забезпечення в режимі портативності (Portability).
3) Наявність інструкції користуванням програмним забезпеченням.
4) Наявність інформації про версію програмного забезпечення та номер її зборки.

3.2 Ескізний проект

3.2.1 Розробка діаграми варіантів використання

Діаграма прецедентів або діаграма використання – це діаграма, яка описує відносини між учасниками і прецедентами в системі. Актори є сутністю, з якими система взаємодіє під час своєї роботи. Кожен актор очікує, що система буде вести себе абсолютно передбачувано без жодних відхилень.

Щоб прояснити функціональність, ідентифікувати схему функціональної структури, ідентифікувати пов'язані модулі і досліджувати динаміку програмного забезпечення в цілому, створюється діаграма прецедентів, яка відображає системні прецеденти (use cases), системне середовище (дійові особи актори – actors) і відносини між ними (діаграми прецедентів – use cases diagrams).

Відобразимо інформаційні потоки між програмним забезпеченням та зовнішніми сутностями, які виникають при виконані певних функцій засновуючись на моделі прецедентів. При описі процесу зазвичай намагаються знайти такі питання:

- перший крок в роботі з програмою?
- як користувач взаємодіє з програмним забезпеченням?
- які завдання може виконувати програма?
- яку інформацію може отримати користувач?
- чи задовольняє функціонал усім вимогам програмного забезпечення?

Рисунок 3.1 – Діаграма прецедентів для розроблюваного ПЗ

Розглянемо більш детально кожний варіант використання, надаючи опис потоків подій.

Специфікація варіанту використання «Створення моделі за допомогою мережі Петрі»

1. Призначення: призначений для створення документа, в якому можна побудувати мережу Петрі.
2. Учасники: активний суб’єкт «Користувач».
3. Передумови: перед активізацією даного варіанта використання повинна бути відкрита програма та користувач повинен бути знайомий з моделюванням методом побудови мереж Петрі.
4. Стартова точка: ініціалізується активним суб’єктом «Користувач».
5. Процедура (сценарій): Користувач обирає команду «New» на панелі інструментів програми, перед ним відкривається чисте вікно, в якому він може графічно зобразити етапи виконання ПЗ, яке планується змоделювати у вигляді позицій та переходів, з’єднаних між собою.
6. Аварійні умови (при яких умовах настає аварійне): немає.
7. Результат: чисте робоче вікно.

8. Умови після закінчення використання: немає.
9. Послідуюче: після виклику варіанта використання «Створення моделі за допомогою мережі Петрі» користувач може продовжувати роботу у тому ж напрямі обравши «Додавання або видалення елементів», «Редагування моделі», «Перегляд інструкції з використання», «Перевірка працездатності».
10. Обмеження / виключення: створення нового документу виконується тільки при відкритому документі.
11. Особливості: немає.
12. Коментарі: немає.
13. Відповідні вимоги: немає.
14. Опис: створення нового документу виконується активним суб’єктом «Користувач» і надає подальшу можливість моделювання проекту.

Специфікація варіанту використання «Редагування моделі»

1. Призначення: призначений для редагування документу, в якому вже побудована мережа Петрі.
2. Учасники: активний суб’єкт «Користувач».
3. Передумови: перед активізацією даного варіанта використання повинна бути відкрита програма, в ній створена модель у вигляді мережі Петрі, яку необхідно відредагувати.
4. Стартова точка: ініціалізується активним суб’єктом «Користувач».
5. Процедура (сценарій): Користувач може відредагувати створену раніше мережу, виправити помилки, що були зроблені при попередній побудові.
6. Аварійні умови (при яких умовах настає аварійне): немає.
7. Результат: відредагована мережа.
8. Умови після закінчення використання: немає.
9. Послідуюче: після виклику варіанта використання «Редагування моделі» користувач може продовжувати роботу у тому ж напрямі

обравши «Додавання або видалення елементів», «Перегляд інструкції з використання», «Перевірка працездатності».

10. Обмеження / виключення: редагування документу можливо лише тоді, коли вже створена модель.
11. Особливості: немає.
12. Коментарі: немає.
13. Відповідні вимоги: немає.
14. Опис: редагування документу виконується активним суб’єктом «Користувач» до тих пір поки модель не буде вдосконалена.

Специфікація варіанту використання «Додавання або видалення елементів»

1. Призначення: призначений для додавання нових позицій та переходів, з’єднання їх між собою, позиції являють собою умови, при яких виконуються дії — переходи.
2. Учасники: активний суб’єкт «Користувач».
3. Передумови: перед активізацією даного варіанта використання повинна бути відкрита програма та користувач повинен бути знайомий з моделюванням методом побудови мереж Петрі.
4. Стартова точка: ініціалізується активним суб’єктом «Користувач».
5. Процедура (сценарій): Користувач обирає необхідну команду на панелі інструментів програми, можна додати нові позиції чи переходи, або видалити їх, у позиції можна додати маркування, а переходам надати пріоритет. Наступним кроком буде з’єднання позицій та переходів, та надання мережі наглядності.
6. Аварійні умови (при яких умовах настає аварійне): немає.
7. Результат: побудована модель у вигляді мережі Петрі.
8. Умови після закінчення використання: немає.
9. Послідуюче: після виклику варіанта використання «Додавання або

видалення елементів» користувач може продовжувати роботу у тому ж напрямі обравши «Редагування моделі», «Перегляд інструкції з використання».

10. Обмеження / виключення: додавання або видалення елементів можливо лише у відкритому вікні програми.
11. Особливості: з’єднувати можна лише позиції з переходами і навпаки, маркування можуть мати лише позиції, а приорітет – переходи.
12. Коментарі: немає.
13. Відповідні вимоги: немає.
14. Опис: додавання та видалення елементів виконується активним суб’єктом «Користувач» і надає подальшу можливість перевірки працездатності моделі.

Специфікація варіанту використання «Аналіз працездатності мережі Петрі»

1. Призначення: призначений для перевірки працездатності мережі, що була побудована.
2. Учасники: активний суб’єкт «Користувач».
3. Передумови: перед активізацією даного варіанта використання повинна бути відкрита програма, в ній створена модель у вигляді мережі Петрі, яку необхідно протестувати на працездатність.
4. Стартова точка: ініціалізується активним суб’єктом «Користувач».
5. Процедура (сценарій): Користувач обирає команду «Run» на панелі інструментів програми на клікає на умову яку необхідно перевірити. Якщо умова здійснюється – відбувається подія. Також замірюється час роботи мережі, і після виконання команди «Stop» – час зупиняється.
6. Аварійні умови (при яких умовах настає аварійне): якщо клікнути на позицію в якій немає фішки — ніякі дії не відбудуться.
7. Результат: мережа перевірена на працездатність.

8. Умови після закінчення використання: якщо фішки повернулися в початковий стан – мережа побудована вірна і даний алгоритм працездатний.
9. Послідуюче: після виклику варіанта використання «Перевірка працездатності мережі» користувач може продовжувати роботу у тому ж напрямі обравши «Аналіз мережі матричним методом», «Аналіз деревом покриваючих маркувань», «Перегляд інструкції з виконання».
10. Обмеження / виключення: перевірити на працездатність можна лише вже створену модель, вона повинна бути замкнена та мати фішки в позиціях.
11. Особливості: немає.
12. Коментарі: немає.
13. Відповідні вимоги: немає.
14. Опис: перевірка працездатності мережі виконується програмою, якщо модель працездатна, можна виконувати її аналіз.

Специфікація варіанту використання «Аналіз мережі матричним методом»

1. Призначення: призначений для перевірки досяжності маркування в мережі.
2. Учасники: активний суб’єкт «Користувач».
3. Передумови: перед активізацією даного варіанта використання повинна бути розроблена мережа, яка вже перевірена на працездатність цією ж програмою.
4. Стартова точка: ініціалізується активним суб’єктом «Користувач».
5. Процедура (сценарій): Користувач обирає команду «Analysis > Matrix» на головній панелі меню програми. Перед користувачем з’являється віконце в якому через пробіл потрібно ввести необхідне досяжне маркування.
6. Аварійні умови (при яких умовах настає аварійне): якщо кількість цифр

не співпадає з кількістю позицій — виникне помилка про невідповідність розмірів векторів.

7. Результат: перелік переходів, які необхідно виконати для отримання заданого маркування, або помилку про неможливість досяжності.
8. Умови після закінчення використання: немає.
9. Послідуюче: після виклику варіанта використання «Аналіз мережі матричним способом» користувач може продовжувати роботу у тому ж напрямі обравши «Аналіз деревом покриваючих маркувань» та «Перегляд інструкції з виконання».
10. Обмеження / виключення: перевірити на працездатність можна лише вже створену робочу модель.
11. Особливості: немає.
12. Коментарі: немає.
13. Відповідні вимоги: немає.
14. Опис: перевірка працездатності мережі виконується програмою, якщо модель працездатна, можна виконувати її матричний аналіз.

Специфікація варіанту використання «Аналіз деревом покриваючих маркувань»

1. Призначення: призначений для перевірки мережі на тупики, обмеженість, безпечність, досяжність та живість.
2. Учасники: активний суб’єкт «Користувач».
3. Передумови: перед активізацією даного варіанта використання повинна бути розроблена мережа, яка вже перевірена на працездатність цією ж програмою.
4. Стартова точка: ініціалізується активним суб’єктом «Користувач».
5. Процедура (сценарій): Користувач обирає команду «Analysis -> Tree» на головній панелі меню програми. Перед користувачем з’являться віконце в якому зображене дерево відповідне до заданої мережі. Нижче

розташована кнопка «Analys», в результаті натиску якої отримаємо результат аналізу.

6. Аварійні умови (при яких умовах настає аварійне): якщо мережа не працездатна – дерево не побудується.
7. Результат: дерево покриваючих маркувань та його аналіз.
8. Умови після закінчення використання: немає.
9. Послідуюче: після виклику варіанта використання «Аналіз деревом покриваючих маркувань» користувач може продовжувати роботу у тому ж напрямі обравши «Аналіз матричним методом» та «Перегляд інструкції з виконання».
10. Обмеження / виключення: перевірити на працездатність можна лише вже створену робочу модель.
11. Особливості: немає.
12. Коментарі: немає.
13. Відповідні вимоги: немає.
14. Опис: перевірка працездатності мережі виконується програмою, якщо модель працездатна, можна побудувати дерево покриваючих маркувань та провести його аналіз.

Специфікація варіанту використання «Перегляд інструкції з виконання»

1. Призначення: призначений для допомоги користувачу в роботі з програмним продуктом.
2. Учасники: активний суб’єкт «Користувач».
3. Передумови: перед активізацією даного варіанта використання повинна бути запущена програма.
4. Стартова точка: ініціалізується активним суб’єктом «Користувач».
5. Процедура (сценарій): Користувач обирає команду «Help» на головній панелі меню програми. Перед користувачем з’являться віконце в якому будуть описані рекомендації щодо користування програмним

продуктом.

6. Аварійні умови (при яких умовах настає аварійне): немає.
7. Результат: допомога користувачеві.
8. Умови після закінчення використання: немає.
9. Послідуюче: після виклику варіанта використання «Перегляд інструкції з використання» користувач може продовжувати роботу з програмою.
10. Обмеження / виключення: немає.
11. Особливості: немає.
12. Коментарі: немає.
13. Відповідні вимоги: немає.
14. Опис: інструкція з використання програмного забезпечення.

3.2.2 Розробка діаграми діяльності

На поточній фазі процесу розробки програмного забезпечення, діаграми діяльності, як правило, створюються для відображення динамічних характеристик системи. У нашому випадку також було б корисно створити діаграму діяльності для деяких випадків використання і для того, щоб конкретно відобразити потік функцій управління, також може скласися така ситуація, що якісь гілки процесу можуть виконуватися паралельно, а також необхідно визначити альтернативний шлях досягнення цілей.

Діаграма діяльностей представлена на рисунках 3.2-3.4.

Рисунок 3.2 – Діаграма діяльності для варіанта використання «Перевірка працездатності мережі»

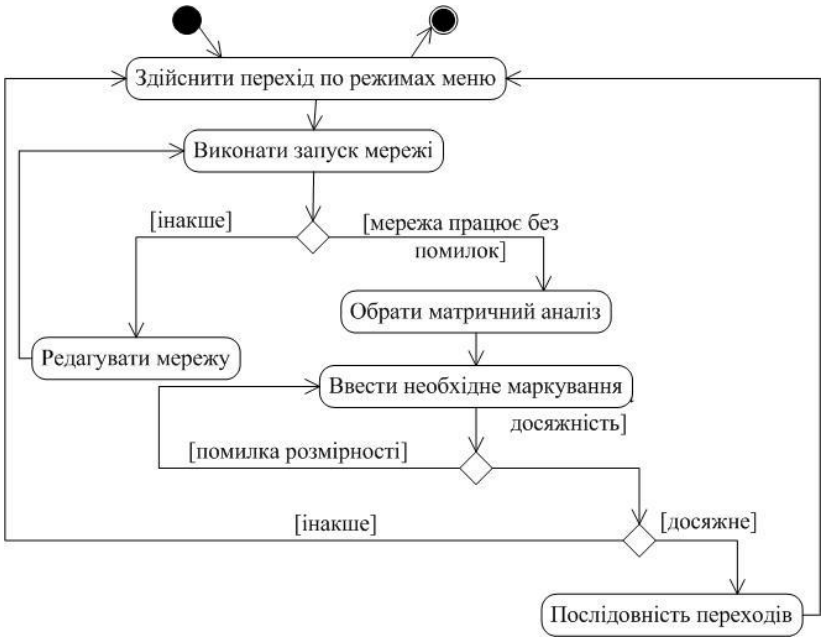


Рисунок 3.3 – Діаграма діяльності для варіанта використання «Аналіз мережі матричним способом»

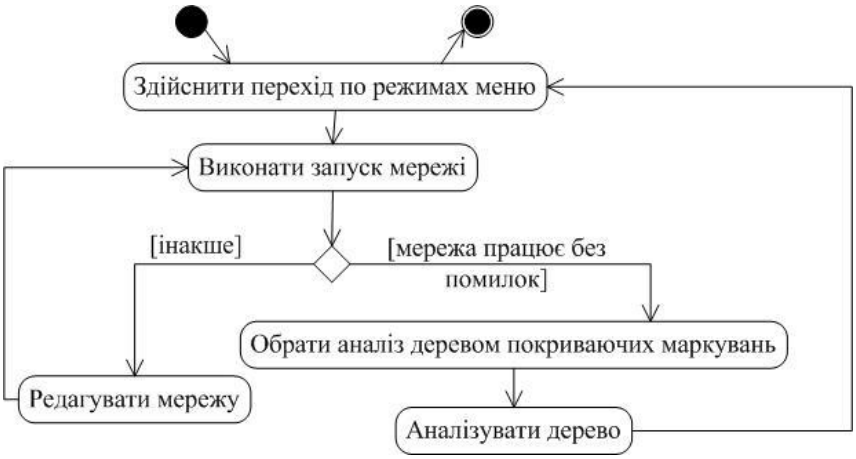


Рисунок 3.4 – Діаграма діяльності для варіанта використання «Аналіз мережі деревом покриваючих маркувань»

3.2.3 Розробка діаграми станів

Сценарії варіантів використання розроблюються для опису поведінки системи, тобто характеристик взаємодії об'єктів в роботі. А іноді необхідно вивчити поведінку окремих об'єктів. Для цього вони використовують діаграми станів, що представляють стан об'єкта, події або повідомлення, які змушують об'єкт переходити з одного стану в інший і дії, що виникають в результаті зміни стану.

У діаграмі станів відображаються всі повідомлення, які людина може отримувати і відправляти. Кожен шлях на діаграмі станів представляє певний сценарій. Аналізуючи поведінку об'єктів, необхідно зосередитися насамперед на питаннях, пов'язаних з тим, що «є» в системі. При цьому аспекти, які зосереджені на тому, «як це відбувається», ігноруються.

Тому постала задача – проаналізувати, що станеться в цій системі, і врахувати всі можливі умови, коли об'єкт (система) виконує певні дії або готується до іншої події.

Діаграма станів в загальному вигляді показана на рисунку 3.5. Є п'ять можливих режимів роботи програми: «Режим редагування», «Режим мережі», «Режим матричного аналізу», «Режим аналізу дерева» і «Інструкції режиму

роботи». Програма має відносно простий графічний інтерфейс користувача і докладні інструкції, тому складності в роботі не повинні виникати.

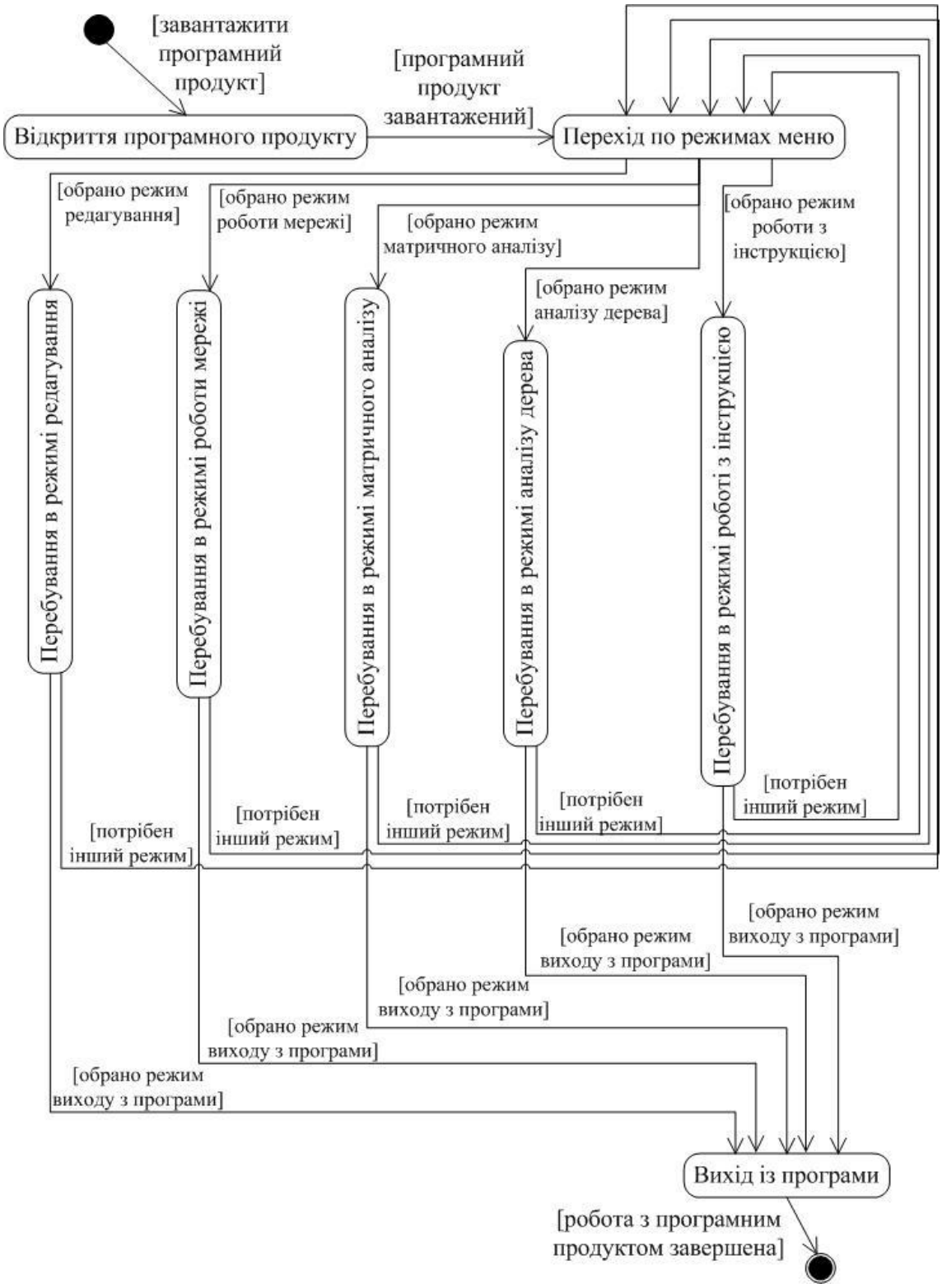


Рисунок 3.5 – Діаграма станів ПЗ контролю якості

Перелік можливих подій в режимах наведені в таблицях 3.1-3.5.

Таблиця 3.1 – Перелік можливих дій в режимі редагування

№	Подія	Дії
1	Відкриття чистої сторінки	Натисніть кнопку «New» на панелі інструментів.
2	Додавання нових позицій в мережу	Натисніть кнопку «Add Place» на панелі та клікніть на місця положення позицій на екрані.
3	Видалення позицій з мережи	Натисніть кнопку «Del Place» на панелі та клікніть на непотрібну позицію.
4	Додавання нових переходів в мережу	Натисніть на кнопку «Add Transition» на панелі та клікніть на місця положення переходів на екрані.
5	Видаленн переходів з мережі	Натисніть кнопку «Del Transition» на панелі та клікніть на непотрібний перехід.
6	Одавання маркувань в позиції	Натисніть кнопку «Add Token» на панелі та клікніть на позиції, в які необхідно додати маркування.
7	Видалення маркувань з позиції	Натисніть кнопку «Del Token» на панелі та клікніть на позицію в якій маркування не потрібні.
8	Пріоритет переходу	Наведіть курсор на перехід та натисніть праву кнопку миші, поставте галочку, або зніміть її, якщо приорітет не потрібен.
9	З’єднання позицій та переходів	Натисніть кноку «Add Arc» на панелі та виберіть позицію і перехід, які необхідно з’єднати.

Продовження таблиці 3.1

1	2	3
10	Переміщення елементів в мережі	Натисніть кнопку «Drag Element» на панелі та переміщуйте елементі перетягуючи їх мишею по екрану, зв’зки зберігаються між елементами.
11	Збереження відредагованого документу	Виберіть на головній панелі меню «File → Save».
12	Відкриття документу	Виберіть на головній панелі меню «File → Open» та кікніть на потрібному документі.

Таблиця 3.2 – Перелік можливих дій програми в режимі роботи мережі

№	Подія	Дії
1.	Початок роботи мережі	Натисніть кнопку «Start» на панелі інструментів програми та клікніть на позицію з якої необхідно почати дії, в ній повинна міститися фішка.
2.	Пріоритет переходу	Наведіть курсор на перехід та натисніть праву кнопку миші, поставте галочку, або зніміть її, якщо приорітет не потрібен.
3.	Завершення роботи	Натисніть кнопку «Stop» на панелі інструментів програми.
4.	Таймер	У лівому верхньому куті вікна рахується час роботи мережі, таймер починає працювати після натиску кнопки «Start» та припиняє після натиску «Stop».
5	Вихід із програми	Виберіть на головній панелі меню «File → Exit».

Таблиця 3.3 – Перелік можливих дій програми в режимі матричного аналізу

№	Подія	Дії
1	Віконання матричного аналізу	Виберіть на головній панелі меню «Analysis → Matrix», у віконці введіть необхідне маркування та натисніть кнопку «ОК»
2	Результат матричного аналізу	Після натиску кнопки «ОК» з'явиться повідомлення про помилку або про досяжність маркування. Якщо воно досяжне то буде зазначений перелік переходів що необхідно виконати.
3	Вихід із програми	Виберіть на головній панелі меню «File → Exit».

Таблиця 3.4 – Перелік можливих дій програми в режимі аналізу дерева

№	Подія	Дії
1	Побудова дерева покриваючих маркувань	Виберіть на головній панелі меню «Analysis → Tree», відкриється нове вікно з побудованим деревом.
2	Аналіз дерева	Після натиску кнопки «Analys» нижче під побудованим деревом з'являться результати аналізу. Аналіз мережі проводиться на пошук тупиків, обмеженість, безпечність, досяжність та живість.
3	Вихід із програми	Виберіть на головній панелі меню «File → Exit».

Таблиця 3.5 – Перелік можливих дій програми в режимі роботи з інструкцією

№	Подія	Дії
1	Виклик справки програми	Виберіть на головній панелі меню «Help».
2	Вихід із програми	Виберіть на головній панелі меню «File → Exit».

3.2.4 Розробка інформаційної моделі

Для розробки інформаційної моделі використовується діаграми классів.

Діаграма класів являє собою набір статичних декларативних елементів моделі. Діаграми класів також можуть бути використані при безпосередньому проектуванні, тобто при розробці нової системи та в зворотній інженерії при описі існуючих та використаних систем. Інформація про діаграму класу безпосередньо відображається в вихідному коді програми. Більшість існуючих інструментів дозволяють моделювати UML з генерацією коду для певної мови програмування. Таким чином, діаграма класів є кінцевим результатом дизайну та відправною точкою процесу розробки.

UML використовується для позначення групи об'єктів, які мають однакову структуру, властивості, поведінку та відносини з об'єктами в інших класах.

Клас – це модель для створення об'єктів. Кожен об'єкт є екземпляром певного класу.

Клас представляє собою шаблон для створення об’єктів. Кожен об’єкт є екземпляром конкретного класу.

Діаграма класів представлена в додатку А. У зв’язку з великою кількістю атрибутів деяких класів на головній діаграмі зобразимо весь перелік та зв’язки. Класи, які мають значну кількість атрибутів, а саме «PetriController», «MatrixController», «ToolBar», «TreeController», «PlaceController»,

«TransitionController», «PetriView», «MatrixView», «TreeView» винесемо окремо, рисунки 3.6-3.13.

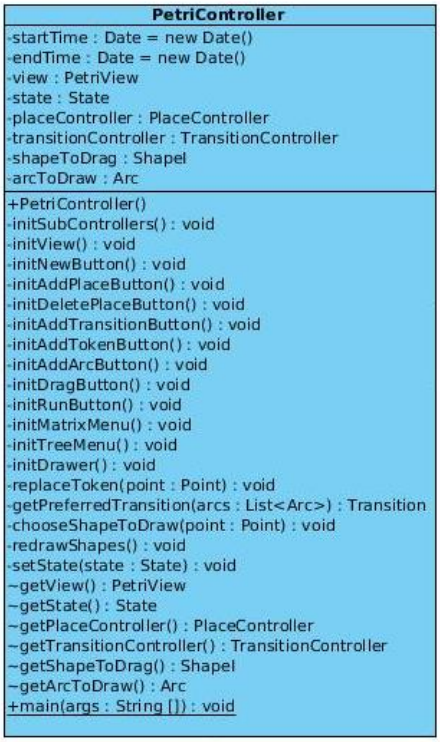


Рисунок 3.6 – Атрибути класу «PetriController»

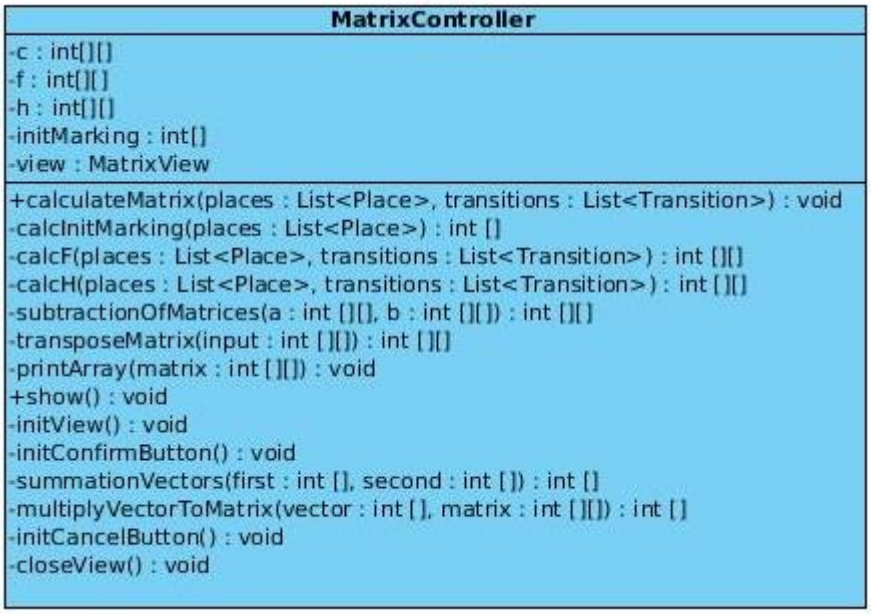


Рисунок 3.7 – Атрибути класу «MatrixController»

Рисунок 3.8 – Атрибути класу «ToolBar»

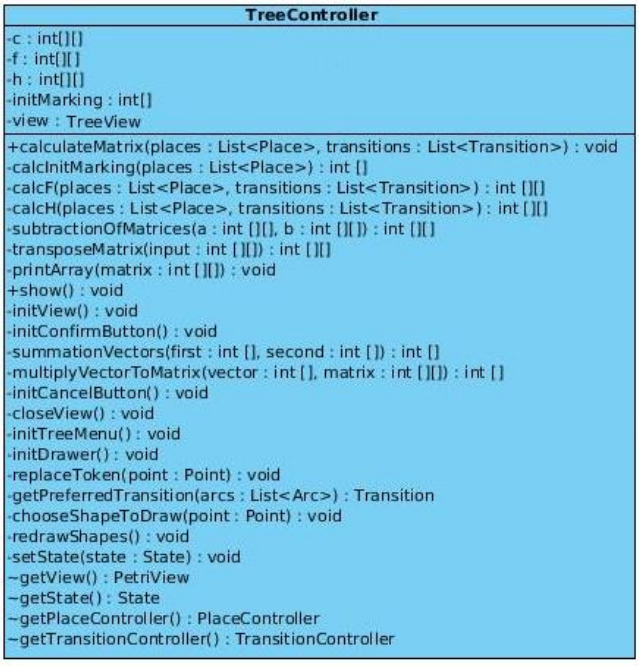


Рисунок 3.9 – Атрибути класу «TreeController»

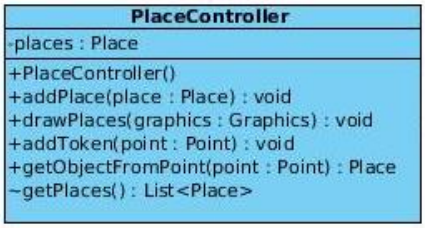


Рисунок 3.10 – Атрибути класу «PlaceController»

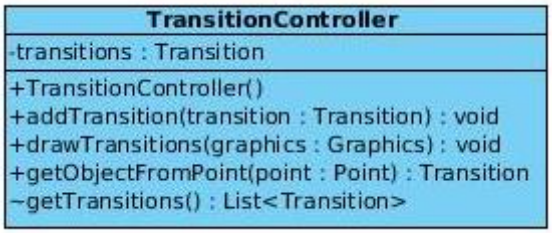


Рисунок 3.11 – Атрибути класу «TransitionController»

Рисунок 3.12 – Атрибути класу «PetriView»

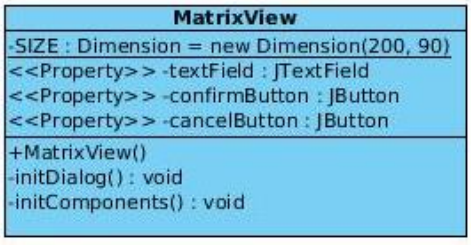


Рисунок 3.13 – Атрибути класу «MatrixView»

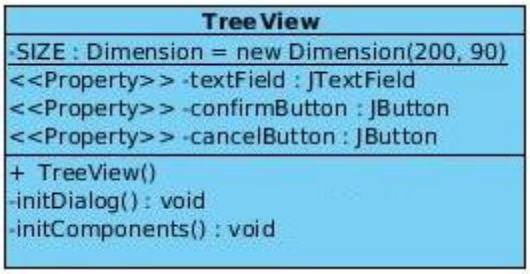


Рисунок 3.14 – Атрибути класу «TreeView»

3.2.5 Моделювання графічного інтерфейсу користувача

Схематично інтерфейс представлено на рисунку 3.15. Також в кінці розділу наведені схематичні зображення інтерфейсів на рисунках 3.16-3.20.

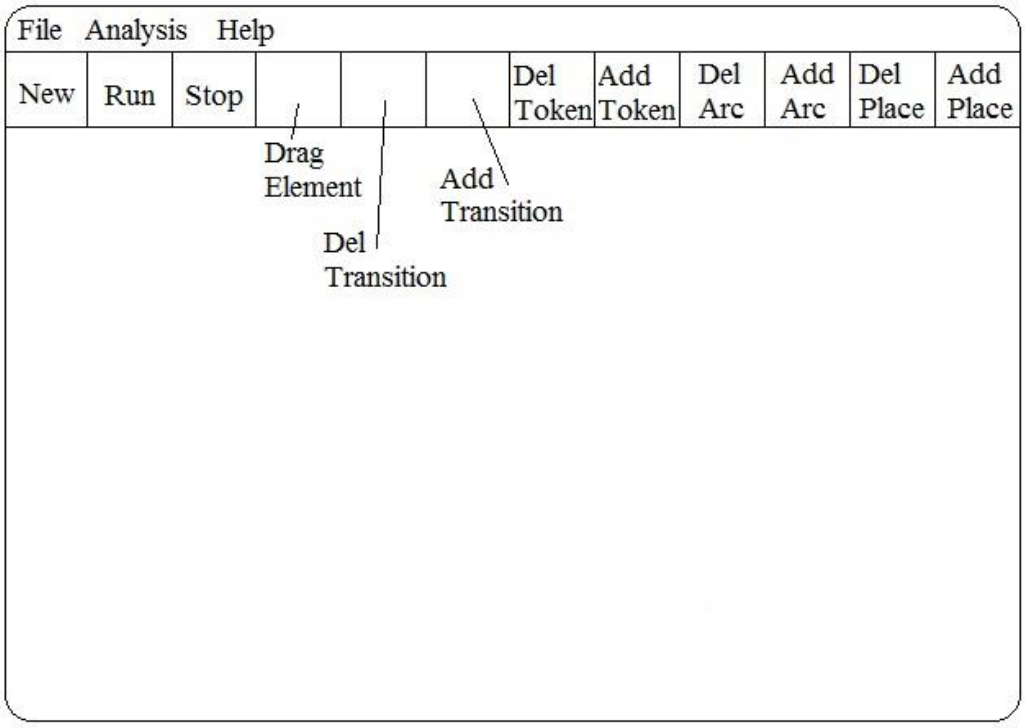


Рисунок 3.15 – Інтерфейс користувача

Графічний інтерфейс користувача доволі простий. Головне меню складається з трьох команд:

Перша команда «File» (рисунок 3.16) – до неї входить створення нового документу, його збереження, відкриття створеного раніше та вихід із програми.

«File → New» – необхідний для створення нового документу, в якому користувач зможе побудувати модель використовуючи мережі Петрі. Модель будується з допомогою клавіш на панелі інструментів:

1. «Add Place» – необхідна для додавання позицій в мережу, що створюється. Позиція – це певна умова, в якій виконуються певні дії.
2. «Del Place» – необхідна для видалення позицій із мережі.
3. «Add Token» – необхідна для додавання маркувань в позиції. Наявність маркування означає виконання умови і, що може бути виконана наступна дія, якщо усі інші умови були виконані.
4. «Del Token» – видаляє маркування із позицій.
5. «Add Transition» – необхідна для додавання в мережу переходів – дій, які виконуються при обставинах, які формуються користувачем.
6. «Del Transition» – необхідна для видалення зайвих переходів з мережі.
7. «Add Arc» – необхідна для сполучення позицій та переходів, визначає послідовність виконання. Однакові сутності не з’єднуються.
8. «Del Arc» – використовується для видалення з’єднань.
9. «Drag Element» – необхідна для пересування елементів в межах робочої області програмного забезпечення, щоб мережа мала логічний вигляд, також, важливо, щоб дуги не перетинались між собою – це необхідно, щоб уникнути плутанини при візуальному ознайомлені побудованої мережі Петрі.
10. «Run» – необхідна для запуску роботи мережі. Під час цієї операції працює таймер, що засікає час роботи мережі Петрі.
11. «Stop» – зупиняє роботу мережі та таймер.

«File → Save» – необхідний для збереження документу(файлу), при обранні цього методу треба вказати шлях збереження.

«File → Open» – необхідний для відкриття збереженого документу.

«File → Exit» – закриває програмне забезпечення.

Друга команда «Analysis» складається з двох видів аналізуи(рисунок 3.17) :

«Analysis → Matrix» – необхідний для матричного аналізу мережі, під час якого перевіряється досяжність того чи іншого маркування. В результаті цього аналізу ви можете визначити порядок переходів для досягнення цього маркування. Схематично вікно матричного аналізу показано на рисунку 3.20.

«Analysis → Tree» – необхідний для побудови дерева покриваючих маркувань та його аналізу. Аналіз визначає ситуації блокування, мережа приймає маркування і зупиняє мережу. Встановлюється статус безпечності мережі, чи є можливим ситуація коли місце розташування містить більше однієї фішки. Перевіряється лімітність існування цілого числа k, що кількість фішок у будь-якій позиції може перевищувати k. Перевіряється, чи мережа створює або породжує фішки. Зображено схематичне вікно для цього аналізу на рисунку 3.20.

Остання третя команда «Help» (рисунок 3.19) – містить коротку інформацію про програмне забезпечення та інструкція того, як будувати мережі Петрі. Також в ній описані відомості про автора програми, дату випуску та можливі оновлення.

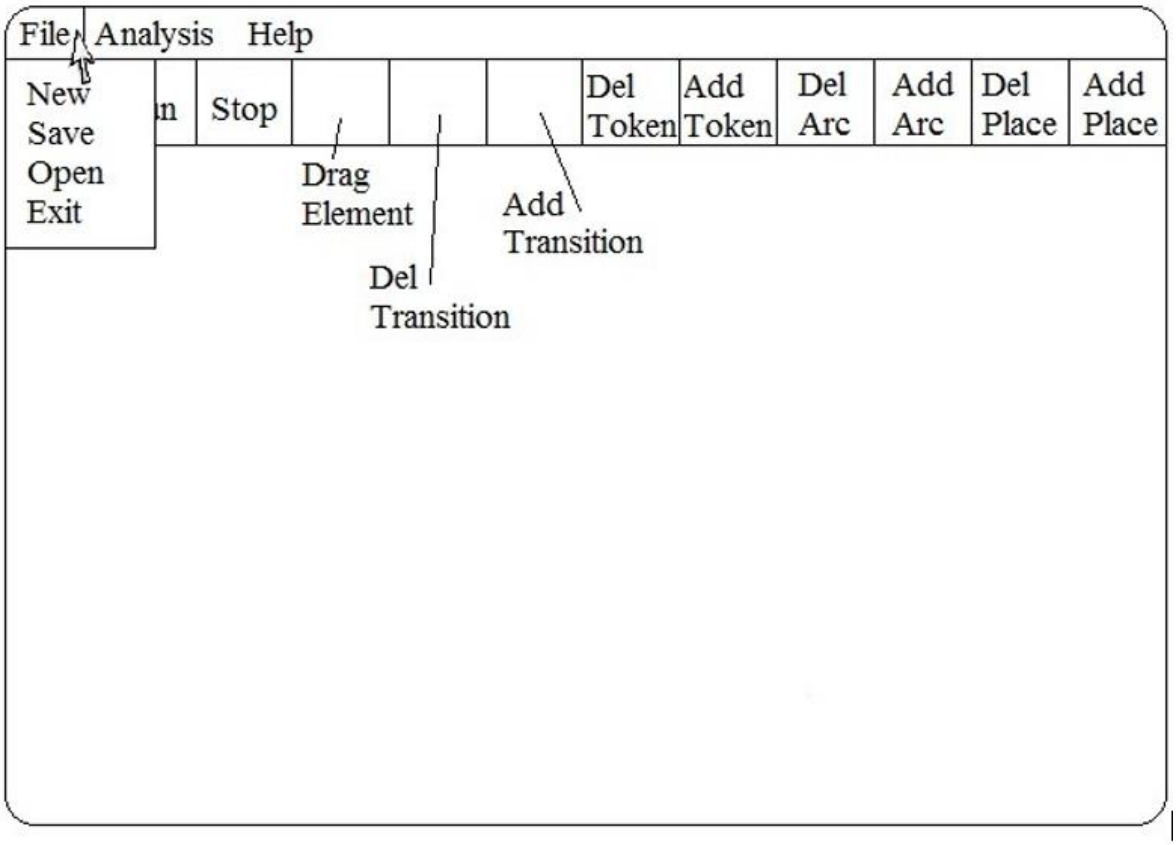


Рисунок 3.16 – Інтерфейс команди «File»

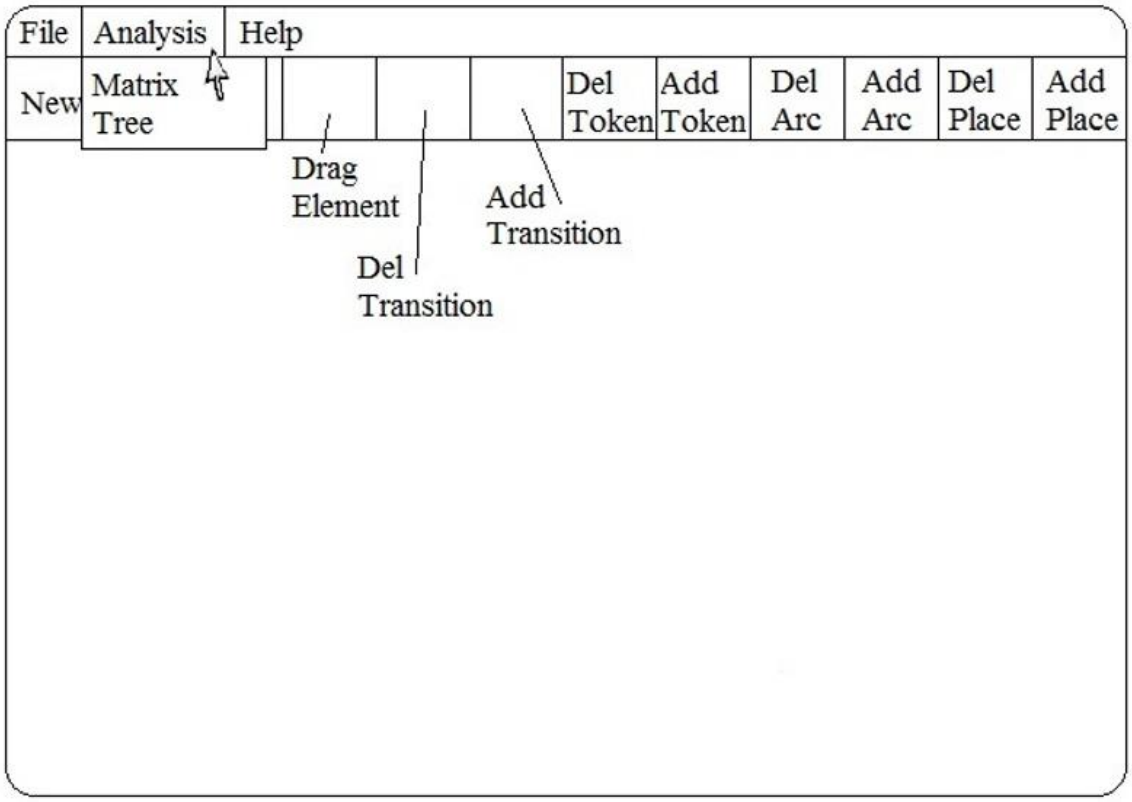


Рисунок 3.17 – Інтерфейс команди «Analysis»

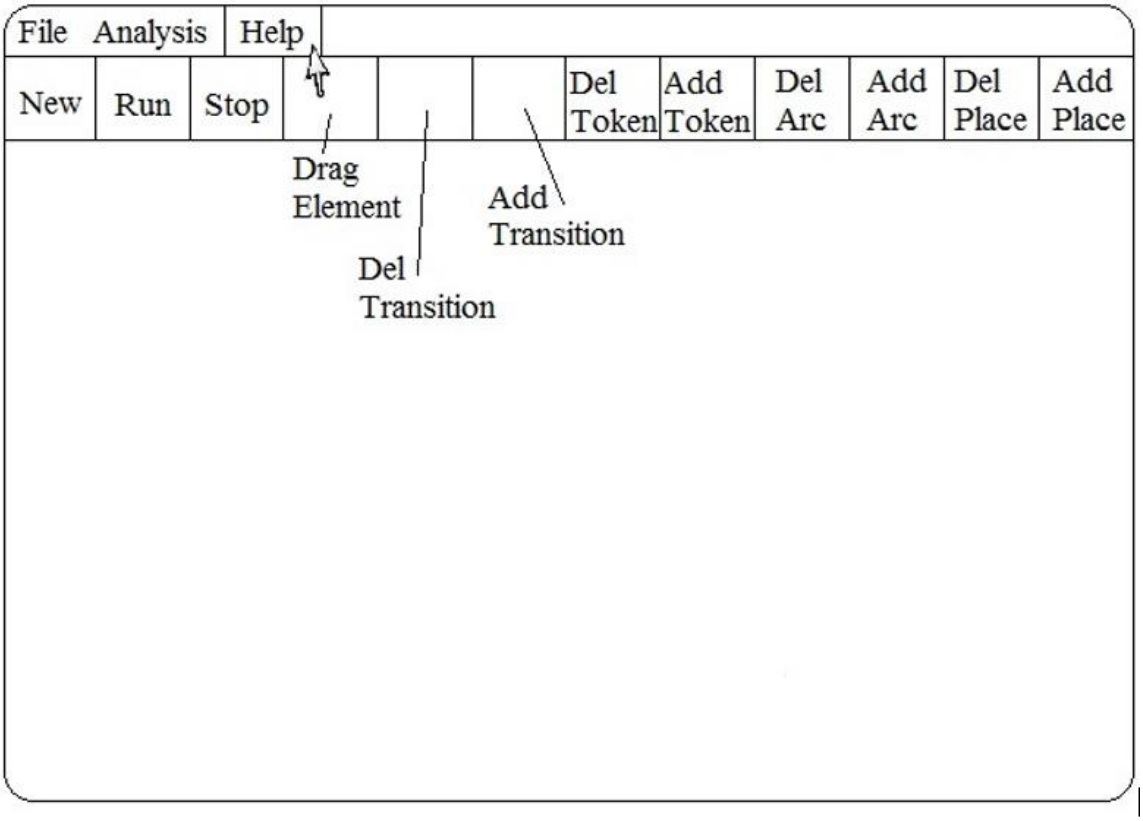


Рисунок 3.18 – Інтерфейс команди «Help»

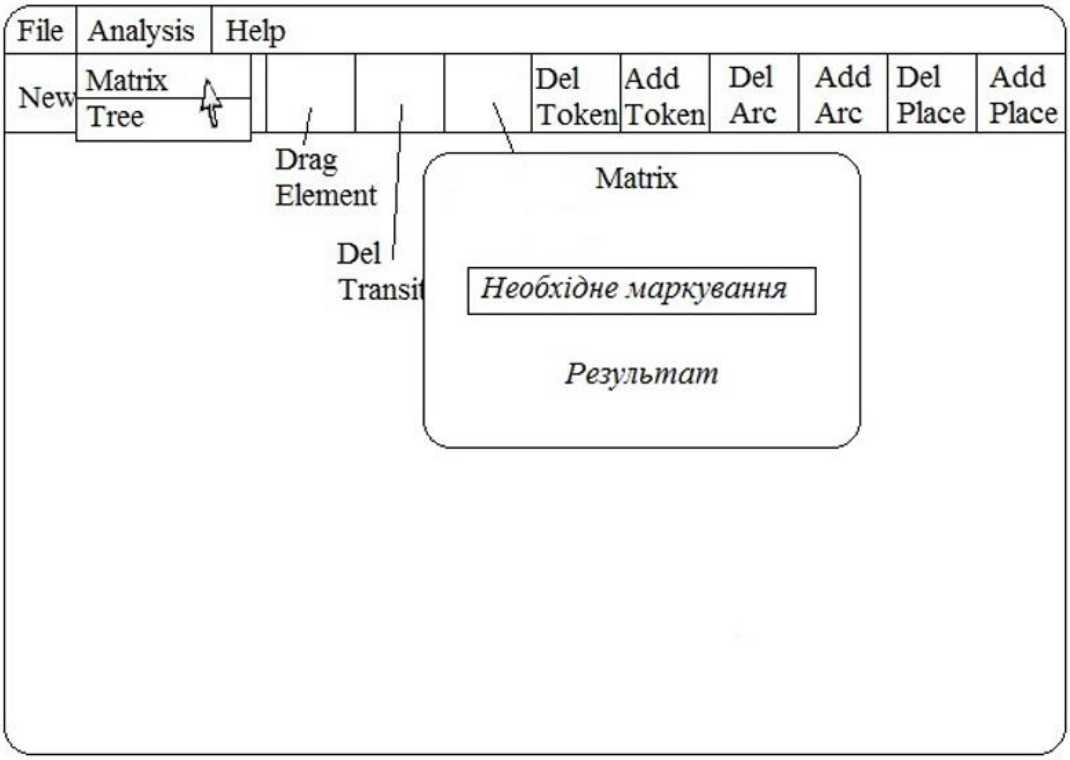


Рисунок 3.19 – Інтерфейс вікна матричного аналізу

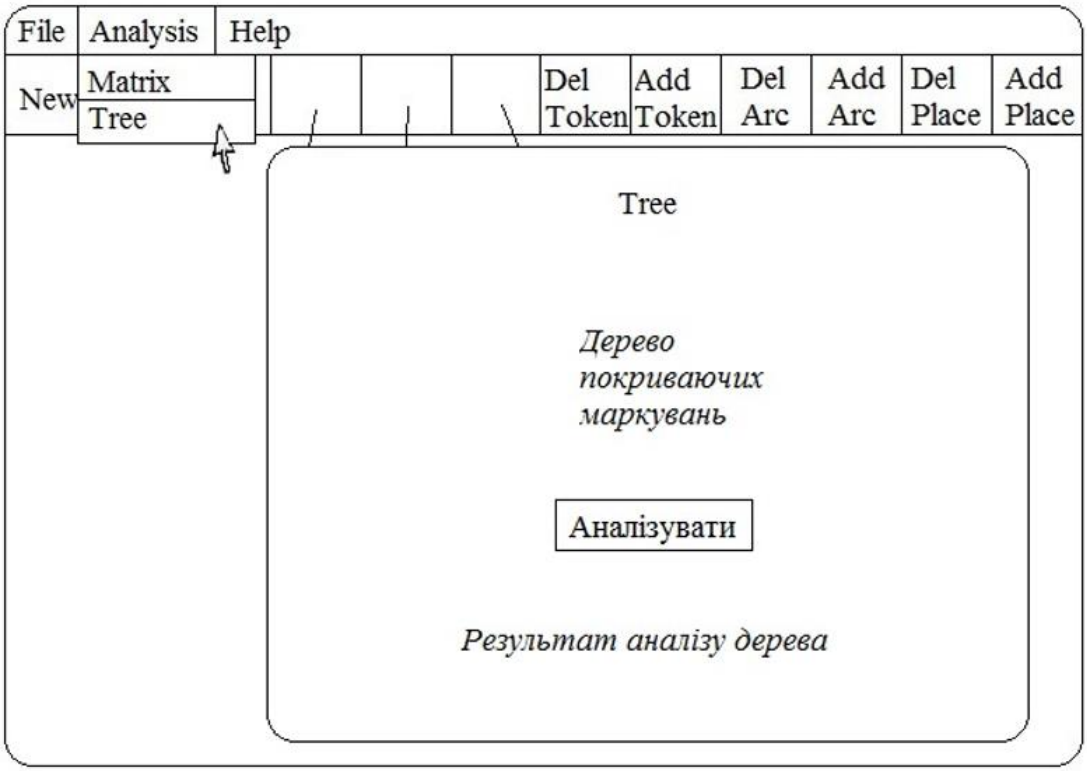


Рисунок 3.20 – Інтерфейс вікна аналізу дерева

3.3 Технічний проект

3.3.1 Розробка статичної моделі

Статична модель представлена у формі діаграми класів та їх специфікацій. Специфікація класу включає наступну інформацію:

- ім’я;
- відповідальність (призначення) класу;
- атрибути (властивості).

В попередньому розділі вже побудована діаграма класів (рисунки 3.6 – 3.15) та були представлені об’єкти, їх зв'язок один з одним. Але не було опису класів та їх призначення. Нижче приведені дані, щодо основних складових.

Специфікації по кожному класу представлені в таблицях 3.6 – 3.20.

Таблиця 3.6 – Специфікація класу «ToolBar»

Складова	Опис
Ім’я	ToolBar
Призначення	Призначений для описання функціонування панелі інструментів та роботи меню.
Атрибути	Створення нового документу, збереження, відкриття вихід із програми, матричний аналіз, побудова дерева, інструкція користувачу, додавання / видалення елементів, зв’язування елементів, переміщення елементів, запуск роботи мережі, зупинка роботи мережі.

Таблиця 3.6 – Специфікація класу «Place»

Складова	Опис
Ім’я	Place
Призначення	Призначений для визначення положення позиції, тобто визначає координати, які передав користувач кліком миші. В цьому місці рисується позиція у вигляді кола, або видаляється, в залежності від того яка дія необхідна.
Атрибути	-

Таблиця 3.7 – Специфікація класу «Token»

Складова	Опис
Ім’я	Token
Призначення	Призначений для додавання або видалення маркування в позиції, яку обрав користувач. Маркування зображується у вигляді невеликого кола.

Продовження таблиці 3.8

1	2
Атрибути	-

Таблиця 3.8 – Специфікація класу «Transition»

Складова	Опис
Ім’я	Transition
Призначення	Призначений для додавання переходів в мережу, по координатах, які визначив користувач. А також їх видалення. Зображується у вигляді прямокутника.
Атрибути	-

Таблиця 3.9 – Специфікація класу «Arc»

Складова	Опис
Ім’я	Arc
Призначення	Призначений для зв’язування позицій та переходів. Слідкує за тим в якому напрямі вони зв’язуються, та не дає можливості зв’язати однакові елементи послідовно.
Атрибути	Координати позицій, координати переходів, напрямок.

Таблиця 3.10 – Специфікація класу «TransitionController»

Складова	Опис
Ім’я	TransitionController
Призначення	Призначений для контролю вірного виконання переходу, та слідкує за кількістю пройдених фішок.
Атрибути	Перехід.

Таблиця 3.11 – Специфікація класу «PetriController»

Складова	Опис
Ім’я	PetriController
Призначення	Призначений для контролю роботи всієї схеми. Це основний елемент програми, який аналізує роботу, всієї мережі, слідкує за породженням нових маркувань та вірного виконання кожного кроку в мережі.
Атрибути	Позиції та переходи мережі, зв’язки між ними, маркування в позиціях, пріоритет в переходах.

Таблиця 3.12 – Специфікація класу «State»

Складова	Опис
Ім’я	State
Призначення	Призначений для слідкуванням за станом. Коли мережа в режимі редагування, коли в режимі роботи. Визначає, яка кнопка меню активна в даний момент.
Атрибути	-

Таблиця 3.13 – Специфікація класу «PlaceController»

Складова	Опис
Ім’я	PlaceController
Призначення	Призначений для контролю вірного проходження позиції маркуванням.
Атрибути	Позиція.

Таблиця 3.15 – Специфікація класу «MatrixController»

Складова	Опис
1	2

Продовження таблиці 3.15

1	2
Ім’я	MatrixController
Призначення	Призначений для проведення матричного аналізу.
Атрибути	Маркування.

Таблиця 3.16 – Специфікація класу «TreeController»

Складова	Опис
Ім’я	TreeController
Призначення	Призначений для побудови дерева та його аналізу.
Атрибути	Позиції, переходи, маркування, зв’язки між елементами схеми.

Таблиця 3.17 – Специфікація класу «PetriView»

Складова	Опис
Ім’я	PetriView
Призначення	Призначений для графічної зміни стану мереж, після виконання кожної дії.
Атрибути	Позиція, маркування.

Таблиця 3.18 – Специфікація класу «TransitionPreferPopup»

Складова	Опис
Ім’я	TransitionPreferPopup
Призначення	Призначений для виставлення пріоритету переходу.
Атрибути	Перехід

Таблиця 3.19 – Специфікація класу «MatrixView»

Складова	Опис
Ім’я	MatrixView
Складова	Опис
Ім’я	TreeView

Таблиця 3.20 – Специфікація класу «TreeView»

Складова	Опис
Ім’я	TreeView
Призначення	Призначений ля графічного зображення дерева та його аналізу.
Атрибути	-

3.3.2 Розробка динамічної моделі

Щоб імітувати динаміку системи, можна використовувати графіки одного типу, а потім перетворити їх в інший тип, не втрачаючи жодної інформації. Як результат, аспекти динаміки системи будуть краще зрозумілі. Для цього ми використовуємо Microsoft Visio і створюємо діаграми взаємодій.

Діаграми взаємодій є загальним найменуванням кооперативних послідовностей та діаграм. Створюючи діаграму послідовності, ви можете легко перетворити її в кооперативну діаграму, не втрачаючи інформацію.

Діаграми послідовності зосереджуються на тимчасових подій. Вони представляють об'єкти та повідомлення, отримані та передані ними.

Кооперативні діаграми чутливі до структурної організації об'єктів, які отримують або надсилають повідомлення. Спільні діаграми показують об'єкти, посилання між ними та повідомлення, які вони надсилають або отримують.

Динамічна модель представлена кооперативами, які є реалізаціями та відповідають діаграмам взаємодії, представлених на рисунках 3.21 – 3.34.

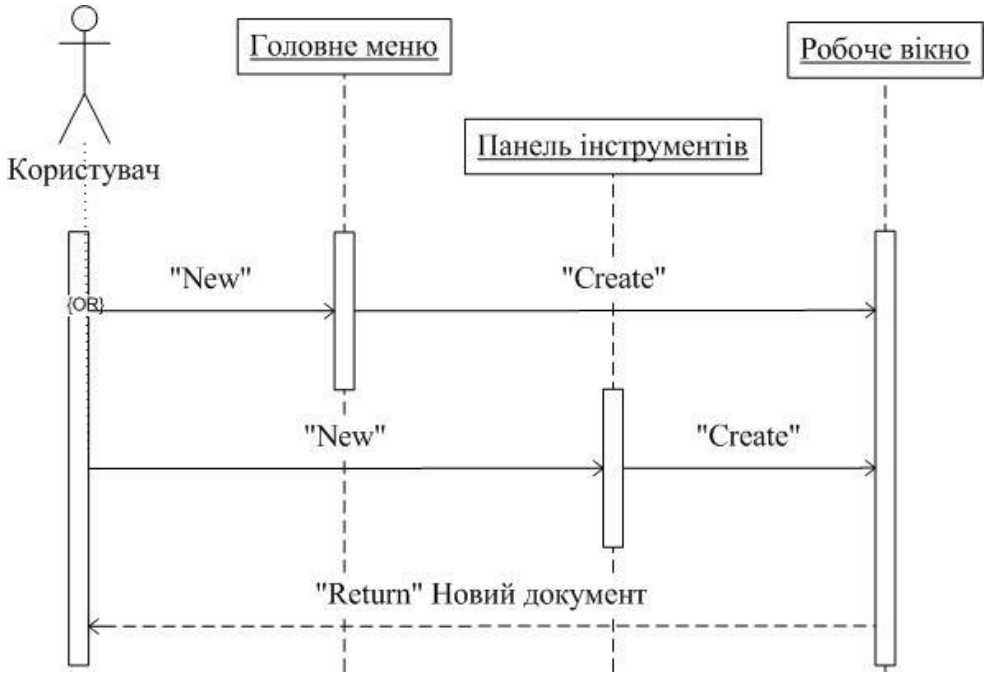


Рисунок 3.21 – Діаграма послідовності ПЗ для варіанту використання «Створення нового документу»

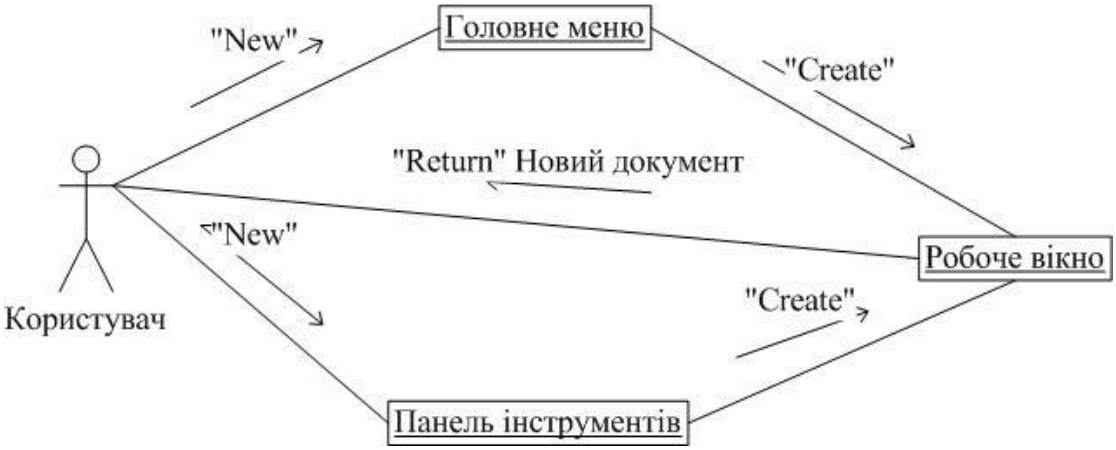


Рисунок 3.22–Діаграма кооперації ПЗ для варіанту використання «Створення нового документу»

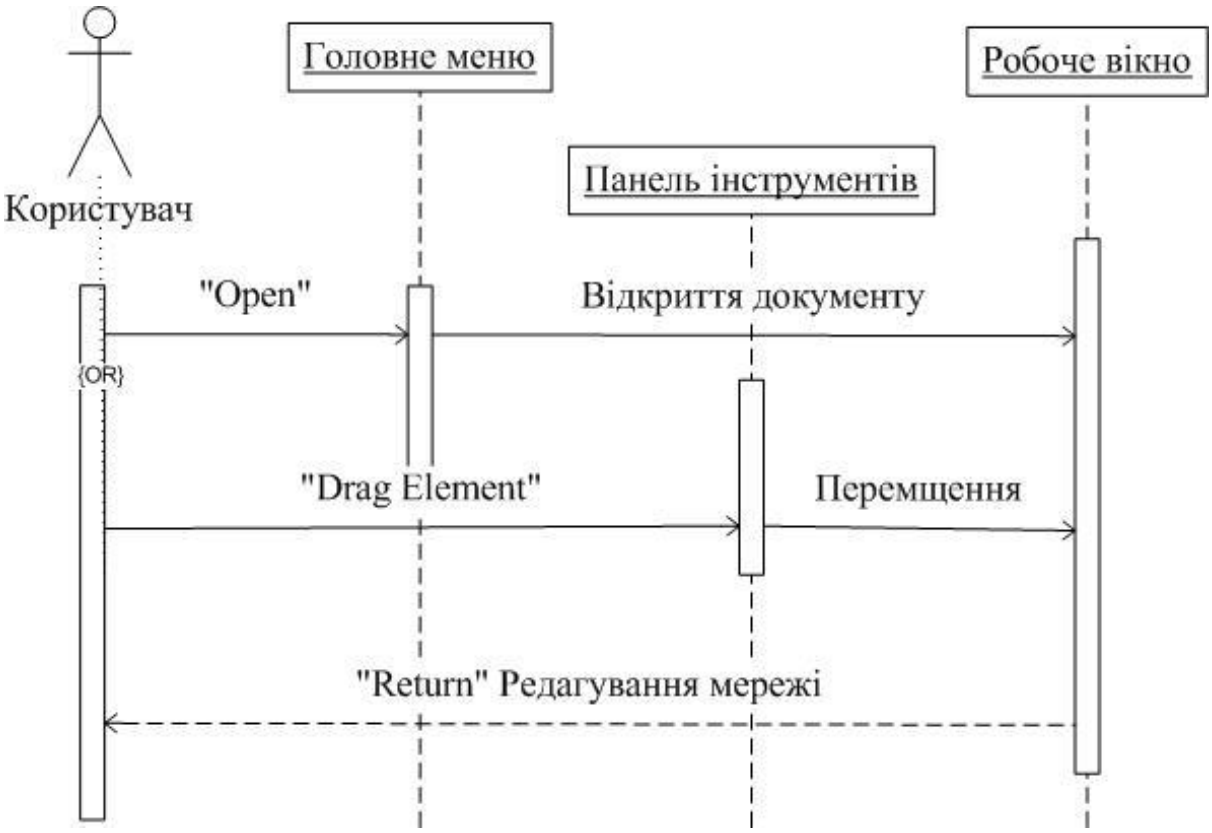


Рисунок 3.23 – Діаграма послідовності ПЗ для варіанту використання «Редагування документу»

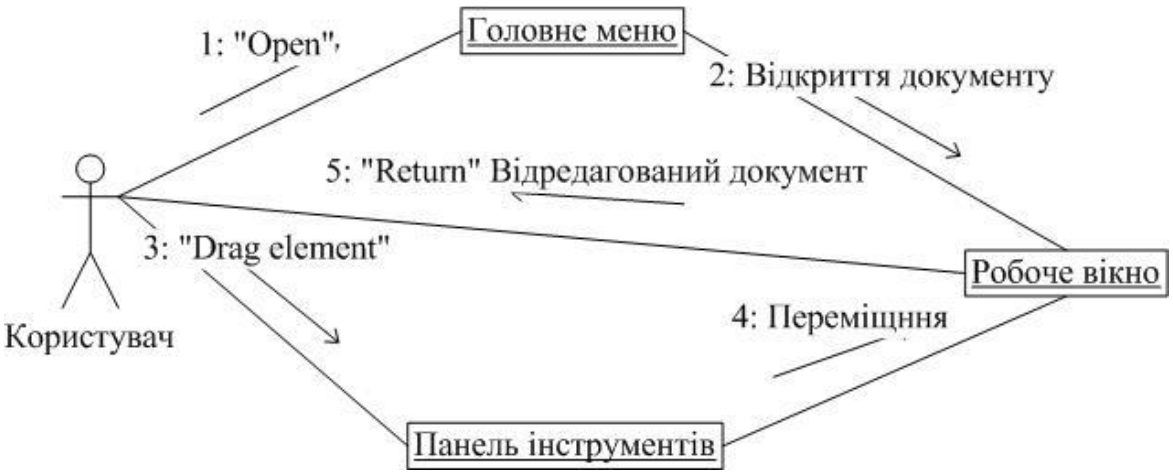


Рисунок 3.24 – Діаграма кооперації ПЗ для варіанту використання «Редагування документу»

Рисунок 3.25 – Діаграма послідовності ПЗ для варіанту використання «Додавання або видалення елементів»

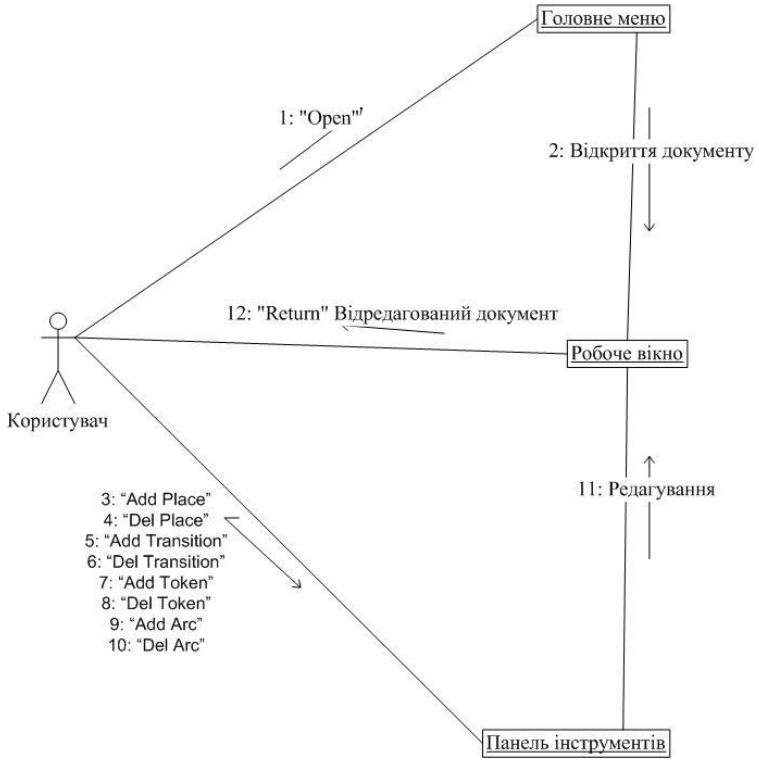


Рисунок 3.26 – Діаграма кооперації ПЗ для варіанту використання «Додавання або видалення елементів»

Рисунок 3.27 – Діаграма послідовності ПЗ для варіанту використання «Перевірка працездатності мережі»

Рисунок 3.28 – Діаграма кооперації ПЗ для варіанту використання «Перевірка працездатності мереж»

Рисунок 3.29 – Діаграма послідовності ПЗ для варіанту використання «Аналіз матричним методом»

Рисунок 3.30 – Діаграма кооперації ПЗ для варіанту використання «Аналіз матричним методом»

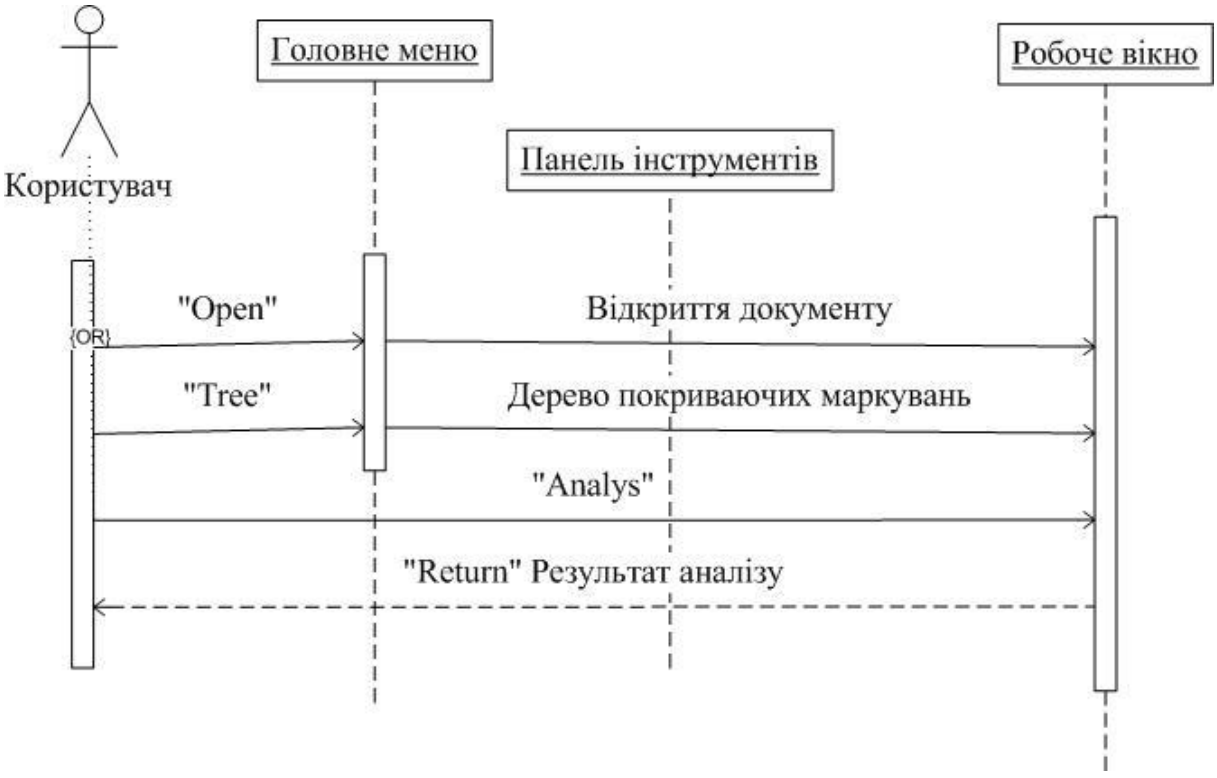


Рисунок 3.31 – Діаграма послідовності ПЗ для варіанту використання «Аналіз деревом покриваючих маркувань»

Рисунок 3.32 – Діаграма кооперації ПЗ для варіанту використання «Аналіз деревом покриваючих маркувань»

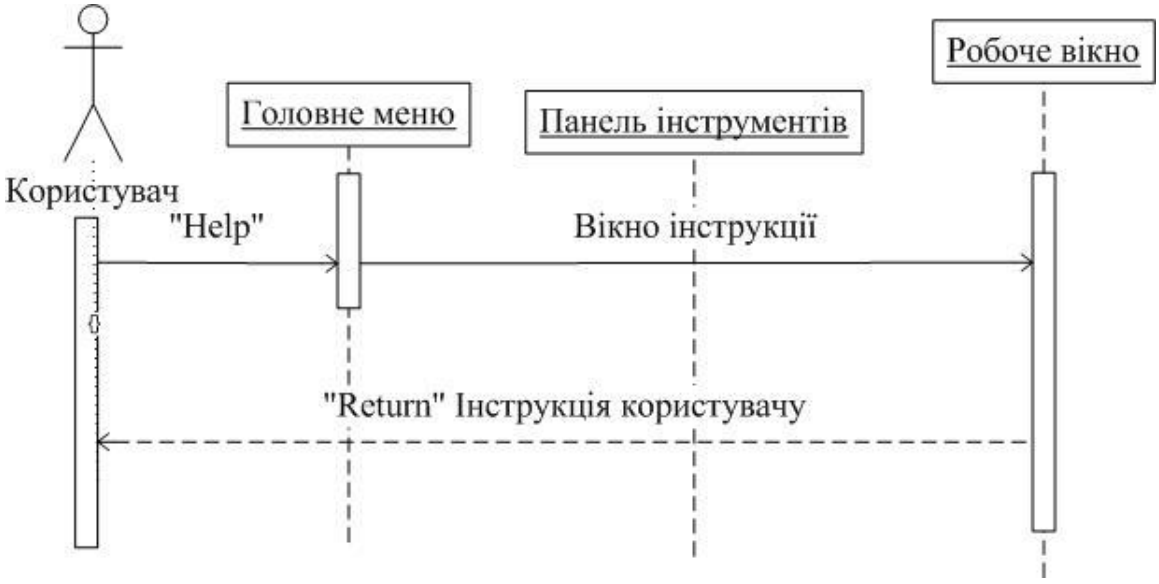


Рисунок 3.33 – Діаграма послідовності ПЗ для варіанту використання «Аналіз деревом покриваючих маркувань»

Рисунок 3.34 – Діаграма кооперації ПЗ для варіанту використання «Аналіз деревом покриваючих маркувань»

3.3.3 Розробка логічної моделі

Логічна модель даних описує поняття ПЗ, їх взаємозв’язки, а також обмеження на дані, накладені ПЗ. На рисунку 3.35 наведена логічна модель структур даних ПЗ контролю якості.

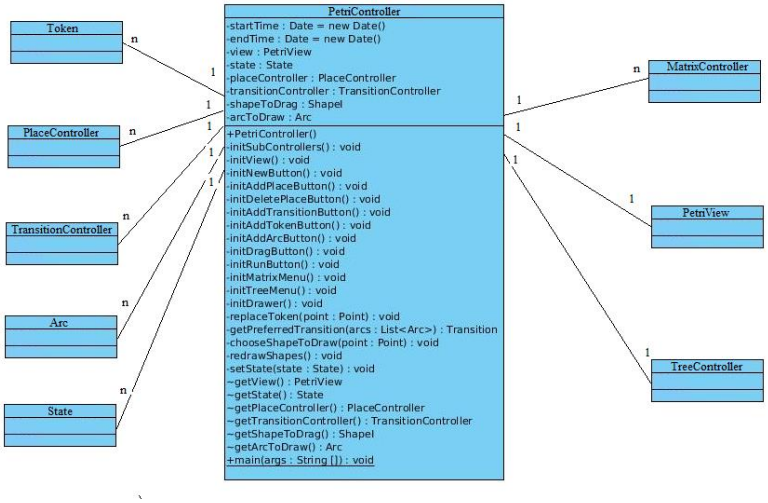


Рисунок 3.35 – Логічна модель структур даних ПЗ контролю якості

3.4 Робочий проект

3.4.1 Вибір мови програмування

Для розробки програмного забезпечення для контролю якості можна використовувати будь-яку об'єктно-орієнтовану мову програмування: C ++, Java та ін. Кожна з них має ряд переваг та недоліків.

Результат аналізу мов програмування, які в теорії можна було використаний для розробки програмного продукту, показав, що більшість мов використовують стандартний набір бібліотек та компонентів. Але найбільш практичною мовою для написання, на мій погляд, була Java.

Програмне забезпечення описане в магістерській дисертації, розроблене в середовищі NetBeans IDE на мові програмування Java. Наступні аспекти призвели до вибору цієї мови програмування та середовища: підтримка методології ООП; кросплатформність; простота розробки графічного інтерфейсу користувача та необхідні структури даних.

Лістинг коду наведено в додатку Б.

3.4.2 Результати розробки

В процесі виконання магістерської роботи було створено програмний продукт, який проводить аналіз побудованої мережі Петрі двома способами: матричним та використовуючи дерево покриваючих маркувань.

Результати роботи програми будуть представлені на основі аудиту контролю якості описаного в другому розділі магістерської роботи.

Опишемо результати роботи програми поетапно на рисунках 3.36-3.40:

1. Треба відкрити «Petri Network» та в ньому створюємо новий документ, вибрав у головному меню «File → New» або на панелі інструментів кнопку «New», (рисунок 3.36).

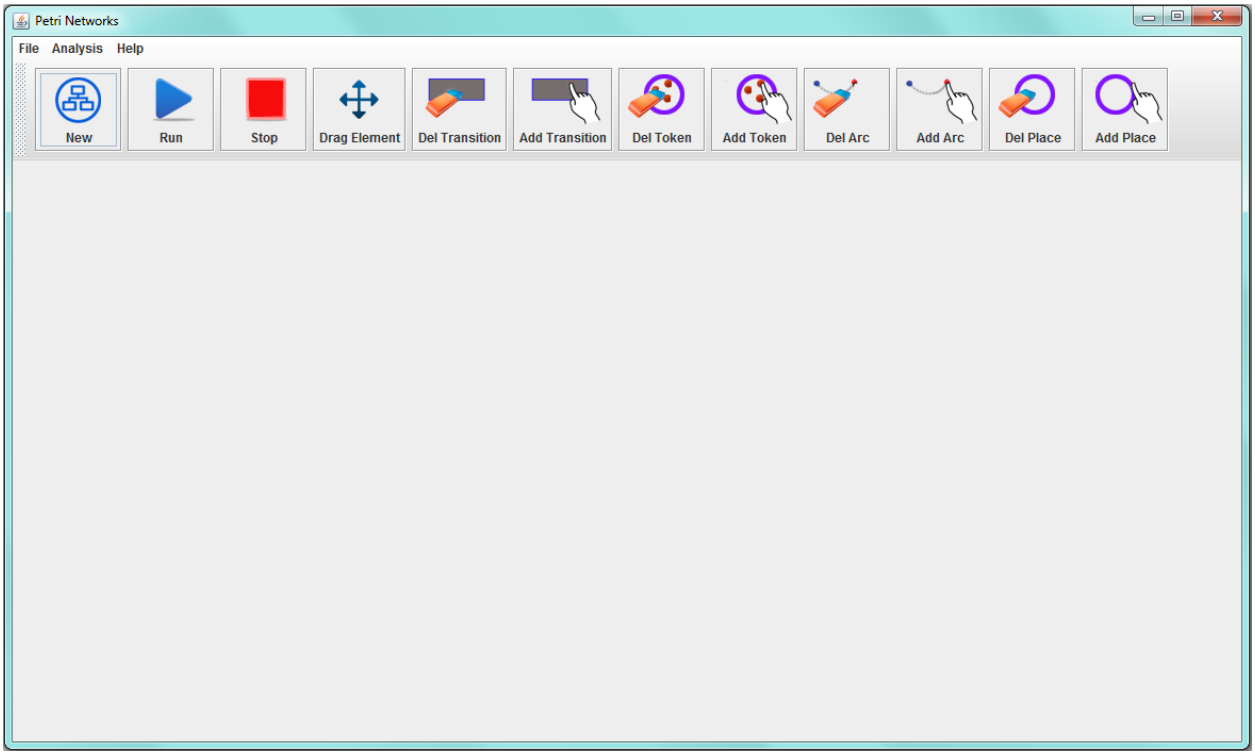


Рисунок 3.36 – Вікно роботи програми «Petri Network» після запуску

2. Ознайомимось з інструкціями користувачу на рисунку 3.37.

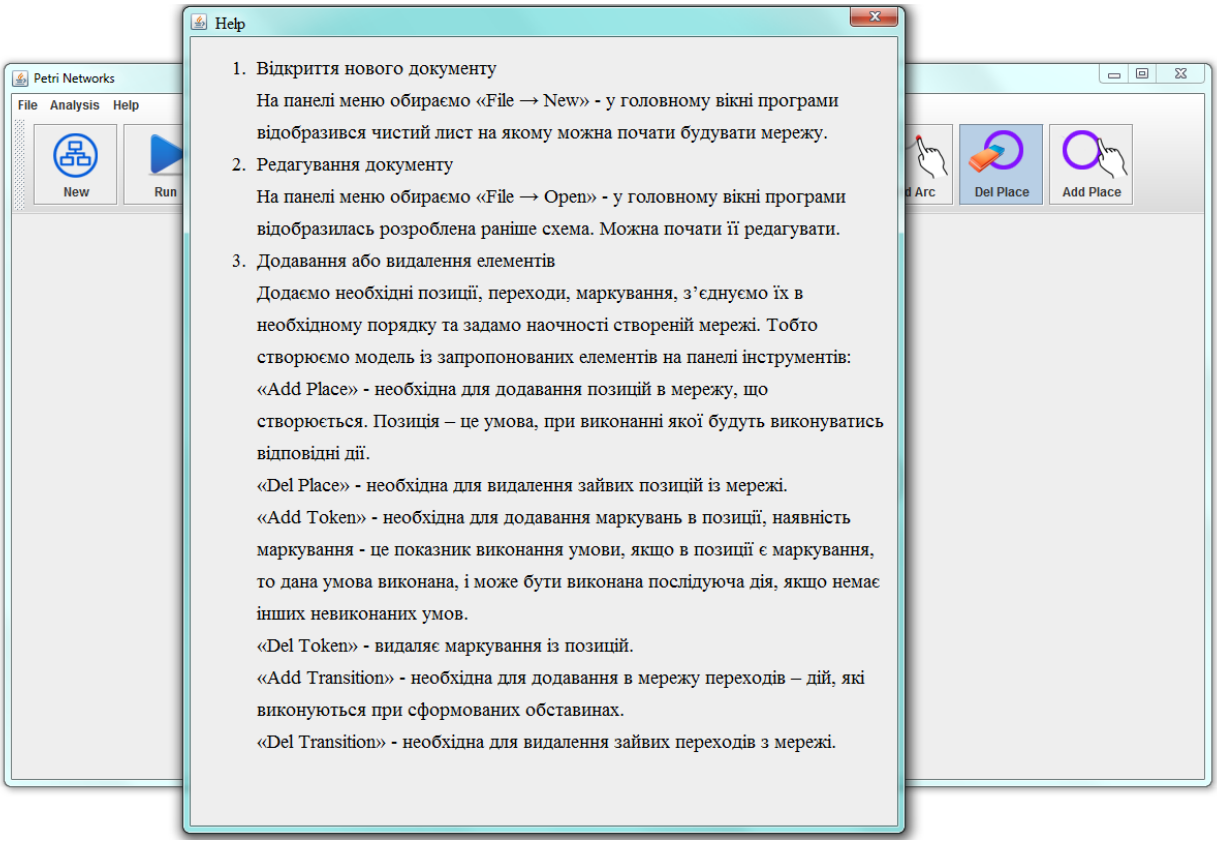


Рисунок 3.37 – Інструкція користувачу

3. Наступним кроком буде побудова моделі процесу контролю якості програмного забезпечення в даній програмі, рисунок 3.38 на основі моделі, що описана у розділі №2.

4. Далі проводиться матричний аналіз для цієї ж моделі, рисунок 3.39, для цього необхідно в головному меню програми обрати «File → Matrix», програма запропонує ввести необхідне маркування, що потрібно досягти, та по натисканню «ОК», отримаємо перелік переходів, що повинні спрацювати для досяжності потрібного маркування.

5. Також необхідно перевірити роботу побудови дерева, для цього в головному меню програми обираємо «File → Tree», отримаємо побудоване дерево покриваючих маркувань. По натисканню «Analysis» програма самостійно проведе аналіз по чотирьом властивостям мережі: обмеженість, живість (активність), досяжність та безпечність, результати цього аналізу представимо на рисунку 3.40.

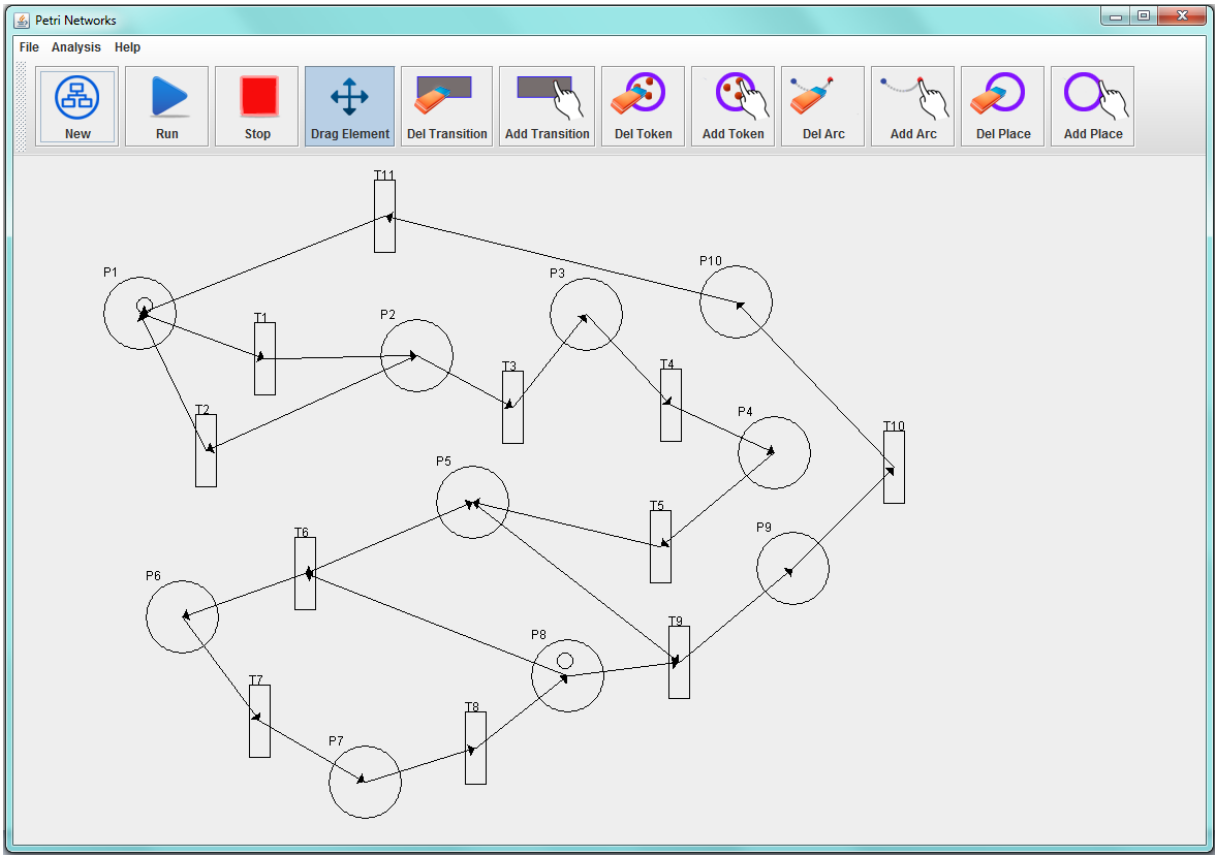


Рисунок 3.38 – Побудована модель контролю якості ПЗ

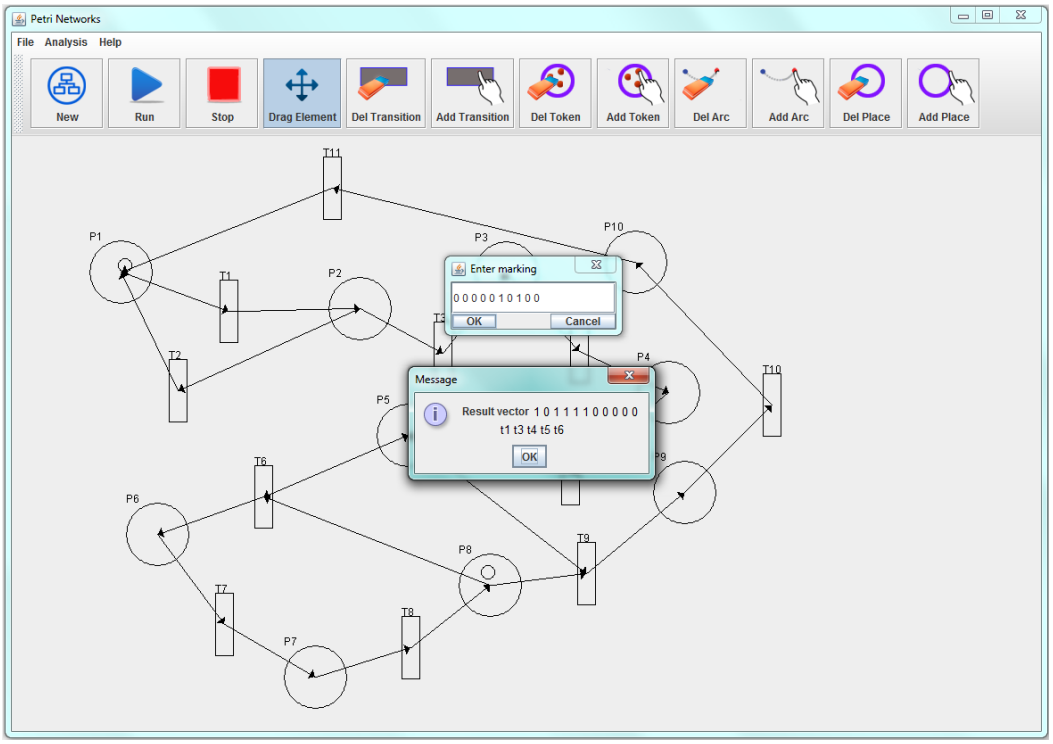


Рисунок 3.39 – Матричний аналіз моделі контролю якості ПЗ

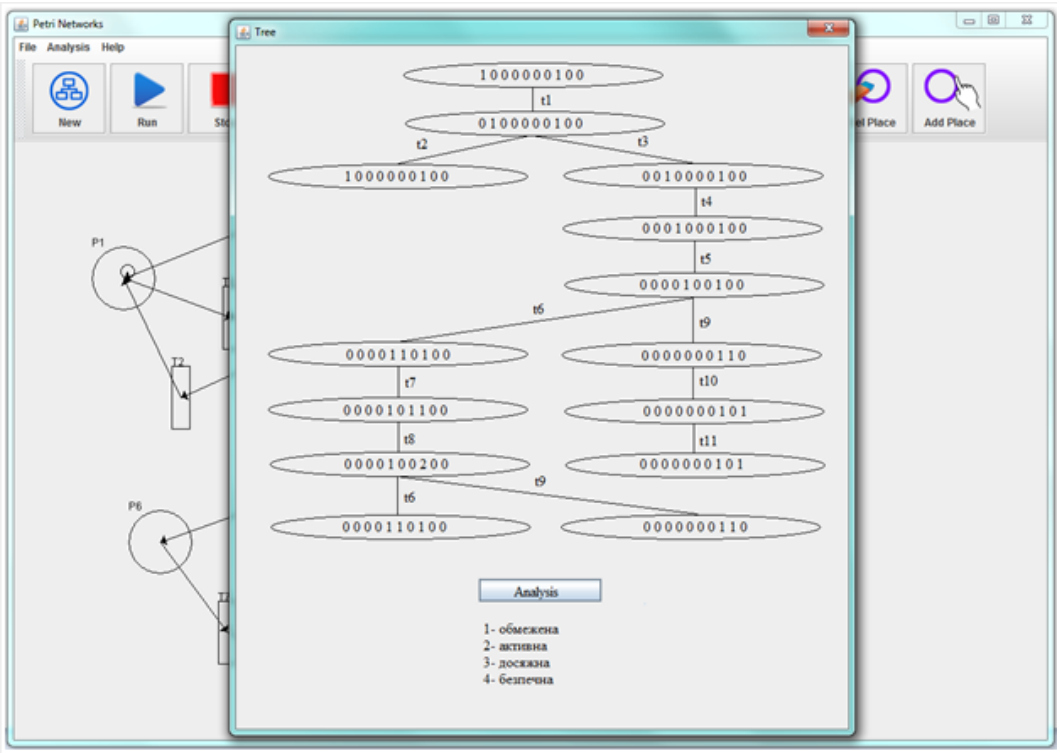


Рисунок 3.40– Дерево покриваючих маркувань для моделі контролю якості ПЗ та його аналіз

3.5 Тестування програмного забезпечення контролю якості

Процедура тестування представляє з себе процедуру експлуатації системи в контрольованих умовах та дозволяє ознайомитись з отриманими результатами. У той же час функціонування системи тестується нормальними та помилковими даними і подіями. Краще всього використовувати метод "знизу вгору" для перевірки розподілених систем. По-перше, розробник повинен вивчити кожен компонент (елемент системи) окремо. Потім перевірити взаємодію компонентів один з одним на автономному комп'ютері. І лише потім перевірити роботу всієї системи в розподіленому середовищі. Всі ці типи тестів виконуються розробником системи або відділом тестування без участі клієнта. Розробнику не потрібен надавати тести та журнали з записами результатів тестування кому-небудь.

Контроль якості програмного забезпечення складається з тестування кожного модуля (режиму) та тестування взаємодії модулів, тобто роботи програмного забезпечення в цілому. Випробування проводяться відповідно функціональних вимог до програмного забезпечення.

Тести проводяться відповідно до двох стратегій.

Перша стратегія під назвою «чорний ящик» – це тест на керування даними або тест введення-виведення. Якщо використовується ця стратегія, програма вважається чорним ящиком. Цей тест призначений для з'ясування обставин, за яких поведінка програми не відповідає її специфікаціям.

Дані тесту використовуються лише відповідно до специфікації програми (тобто без знання внутрішньої структури). У такому підході виявлення всіх помилок є критерієм повного випробування. Даний метод тестування ставить за ціль з’ясувати при яких обставинах програма поводиться не відповідно своїм специфікаціям.

Стратегія «білого ящика», яка керує логікою програми, дозволяє вивчити внутрішню структуру програми. У цьому випадку людина отримує дані тесту

та аналізує логіку програми. Цей метод характеризується виконанням тестів або охоплює логіку програми. Детальний тест білої коробки вимагає виконання кожного шляху програми [16].

Для цієї стратегії існує декілька методів тестування: покриття операторів, покриття рішень, покриття умов, покриття рішень/умов, комбінаторне покриття умов. Для надійного тестування програмного забезпечення треба знати логіку кожної процедури, модуля та перевірити їх взаємодію.

Через розмір програмного коду та складності програмних модулів тестування за допомогою методу "білий ящик" займе багато часу. Тому розроблене програмне забезпечення перевіряється лише методом "чорного ящика".

Використання методу еквівалентності для класифікації хороших і поганих класів. За допомогою цього методу можна виконати досить повну програму тестування. Результати розбивки вхідних і вихідних даних у класи еквівалентності представлені в таблиці. 3.21, а в таблиці 3.21 представлені тести що покривають правильні й неправильні класи еквівалентності [15, 16].

Таблиця 3.21 – Класи еквівалентності

№ класу	Вхідні умови	Правильні класи	Неправильні класи
1	Створення нового документу	Відкриття чистого робочого вікна	-
2	Редагування мережі	Додавання елементів	-
		Видалення елементів	-

Продовження таблиці 3.21

1	2	3	4
2		Наявність пріоритету переходу	Відсутність пріоритету переходу
		Переміщення елементів	-
		Зв’язування елементів різного типу	-
3	Запуск роботи мережі	Перехід маркування в наступне положення	Неможливість виконання переходу
4	Матричний аналіз	Перелік переходів	Неможливість досяжності маркування
		-	Розмірність не співпадає
5	Аналіз дерева	Побудоване дерево та його аналіз	-

Таблиця 3.21 – Тести класів еквівалентності

№ тесту	№ класу	Вхідні дані	Вихідні дані
1	1	Файл нового проекту	Відкриття нового документу
2	2	«Add Place»	Додавання позиції в мережу
3	2	«Del Place»	Видалення позиції з мережі
4	2	Add Token»	Додавання маркування в позиції
5	2	«Del Token»	Видалення маркування з позиції
6	2	«Add Transition»	Додавання переходу в мережу

Продовження таблиці 3.21

1	2	3	4
7	2	«Del Transition»	Видалення переходу з мережі
8	2	Виставлення пріоритету переходу	При виборі переходу – виконається той, що має пріоритет
9	2	Відсутність пріоритету переходу	При виборі з декількох переходів буде виконуватись випадковий перехід
10	2	«Drag Element»	Переміщення елементу по екрану
11	2	«Add Arc»	Зв’язування позицій і переходів
12	3	«Run» та оберіть позицію яка повинна виконатись	Почався відлік часу та маркування перейшло до наступної позиції
13	3	«Run» та оберіть позицію яка не містить фішку	Нічого не відбулось
14	4	Необхідне маркування	Перелік переходів, що повинні відбутись для досяжності цього маркування
15	4	Маркування	Повідомлення про недосяжність цього маркування
16	4	Маркування невірної розмірності	Помилка про невірно введене маркування
17	5	«Tree»	Дерево покриваючих маркувань та результати його аналізу

3.5.1 Випробовування програмного забезпечення

На основі функціональних вимог, що наведені в розділі 3.1.1 було проведено випробування розробленого програмного забезпечення.

1) Відкриття нового документу

Виконані встановлені початкові дії (новий документ обраний). Результат відповідає очікуваному (у головному вікні програми відобразився чистий лист на якому можна почати будувати мережу). Проблем не виникло. Можна починати створювати нову схему.

2) Редагування документу. Виконані встановлені початкові дії (відкрита створена модель). Результат відповідає очікуваному (у головному вікні програми відобразилась створена раніше мережа Петрі). Проблем не виникло. Можна почати редагувати схему.
3) Додавання або видалення елементів. Виконані встановлені початкові дії (перед нами відкрите вікно програми, всі кнопки активні). Додаємо необхідні позиції, переходи, маркування, з’єднуємо їх в необхідному порядку та задамо наочності створеній мережі. Результат відповідає очікуваному (у головному вікні програми відобразилась модель програми у вигляді мережі Петрі). Проблем не виникло. Можна перевіряти працездатність моделі.
4) Перевірка працездатності мережі. Виконані встановлені початкові дії (новий у головному вікні програми відобразилась модель програми у вигляді мережі Петрі). Результат відповідає очікуваному (після натиску кнопки «Run» фішка почала рухатись в заданому напрямку та прийшла в початкове становище). Проблем не виникло. Мережа працездатна.
5) Аналіз мережі матричним методом .Виконані встановлені початкові дії (у головному вікні відкрита працездатна мережа). Результат відповідає очікуваному (після введення необхідного маркування, ми отримали перелік переходів, що потрібно пройти фішці). Проблем не виникло.

6) Аналіз деревом покриваючих маркувань. Виконані встановлені початкові дії (у головному вікні відкрита працездатна мережа). Результат відповідає очікуваному (після обрання в меню аналізу з допомогою дерева покриваючих маркувань ми отримали схематично побудоване дерево та результат його аналізу). Проблем не виникло.

3.6 Висновки

Розроблене програмне забезпечення для побудови математичної моделі процедури проведення аудиту якості, в якому було побудована модель. Модель була перевірена на працездатність та проаналізована двома методами: матричним та по дереву покриваючих маркувань. Дана модель зручна у використанні та доволі універсальна при роботі з нескладними проектами. Вона допомагає визначити помилки та слабкі місця на різних стадіях життєвого циклу проекту, чим мінімізує затрати на виправлення.

Розроблене програмне забезпечення в свою чергу є реалізацією даної моделі, яке визначає час затрачений на реалізацію проекту, може проаналізувати модель розробки нового програмного забезпечення, та ще до початку його реалізації знайти ряд можливих помилок.

Програма має зручний графічний інтерфейс користувача та інструкцію користувача, і не потребує специфічних навиків. Для побудови моделі та проведення її аналізу достатньо прочитати документацію та ознайомитись з теорією мереж Петрі.

4 СТАРТАП-ПРОЕКТ

4.1 Опис ідеї проекту

Сьогодні комп'ютерні інструменти використовуються в повсякденній роботі людей. Комп'ютерні технології дозволяють нам вирішувати цілий ряд завдань. За допомогою них можна автоматизувати та оптимізувати та автоматизувати операції, що доводилось раніше робити вручну або аналізувати складні інженерні розрахунки, які не може виконувати лише одна людина. За допомогою сучасних інструментів програмування є можливість вирішити проблему, не виходячи з дому. Наприклад, такі проблеми, над якими цілі дослідницькі групи працювали в недалекому минулому.

Розвиток та розповсюдження нових технологій призвели до широкого використання комп’ютерів у вирішенні проблем обробки інформації, пов'язаних з виробництвом, контролем та аналізом.

Процес контролю якості програмного проекту – це процес відстеження проміжних результатів проекту, визначення відповідності прийнятим стандартам та розробка заходів для усунення причин відхилень від стандарту. Управління якістю повинно здійснюватися на всіх етапах проекту. Для оцінки відповідності процесів застосовуються критерії для перевірки відповідності попередньої фази. Використання критеріїв означає, що контрольні списки включені до контрольних переліків, узгоджених всіма залученими сторонами.

Таблиця 4.1 – Опис ідеї стартап-проекту

Ідея	Область або галузь застосування	Користь для користувача
Система контролю якості програмного забезпечення	1.IT	Відносна дешевизна, порівняно з існуючими закордонними аналогами.
	2.Менеджмент проектів	Застосування нового підходу до планування та розробки проектів.

Застосування математичної моделі, як основу для програмного забезпечення продукту робить реалізацію більше легкою. Відомо, що в інтернеті існують багато подібних програмних продуктів, але всі вони лише графічні інструменти, для побудови стандартних мереж Петрі.

Аналіз потенційних техніко-економічних переваг ідеї (яка відрізняється від існуючих аналогів і замінників) щодо пропозицій конкурентів полягає в наступному:

- визначення переліку техніко-економічних характеристик і показників ідеї;
- визначення попереднього кола конкурсу (конкурсні проекти) або його замінників або вже присутніх на ринку, а також збір інформації про значення техніко-економічних показників для ідеї проекту і пропонованих проектів; конкурс за списком вище;
- проводиться порівняльний аналіз показників: для власного уявлення визначаються показники з: а) найгіршими значеннями (W, низькими); б) аналогічні значення (N, нейтральний); в) кращі значення (S, сильні) (таблиця 4.2).

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту.

№ п/п	Характеристики ідеї з точки зору економіки та технологічності	Конкуренти				W (слабка сторона)	N (нейтральна	S (сильна сторона)
		Мій проект	GDToolkit	TimeNet	SPNP			
1.	Вартість ПЗ	Низька	Середня	Середня	Висока			+
2.	Доступність	Низький	Низький	Середній	Середній		+	
3.	Кроссплатформність	Так	Так	Так	Ні			+
4.	Підтримка	+	-	+	+		+	

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

4.2 Технологічний проект аудиту

В даному підрозділі було проведено аналіз технологій за допомогою які можна використати в реалізації проекту. Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових (таблиця 4.3):

а) за якою технологією буде виготовлено товар згідно ідеї проекту?

б) чи існують такі технології, чи їх потрібно розробити/доробити?

в) чи доступні такі технології авторам проекту?

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

№ п/п	Суть проекту	Основні використані технології	Наявність в проекті	Доступність технологій
1.	Створення математичної моделі	Мережі Петрі Java	Наявні.	Так.
Обраною технологією реалізації є мережі Петрі, як інструмент, яким зображується математична модель та мова об’єктно-орієнтованого програмування Java, що дозволить зробити програмне забезпечення кросплатформним				

За результатами аналізу таблиці зроблено висновок щодо можливості технологічної реалізації проекту. Технологічним шляхом реалізації проекту було обрано мову об'єктно-орієнтованого програмування Java в середі розробки NetBeans через їх доступність та безкоштовність.

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначивши ринкові можливості, які можуть бути використані для реалізації ринкового проекту та ринкових загроз, які можуть перешкоджати реалізації проекту, можна планувати розробку проекту з урахуванням його стану в умовах ринку, потребам потенційних споживачів та пропозиції. для конкуруючих проектів. Спочатку було проведено відповідний аналіз: наявність попиту, обсяг, динаміка розвитку ринку. Все це наведено в таблиці 4.4.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники, що демонструють стан ринку	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	100000
3	Динаміка ринку(якісна оцінка)	Стагнує
4	Наявність обмежень для входу	-
5	Специфічні вимоги до стандартизації та сертифікації	-
6	Середня норма рентабельності в галузі(або по ринку), %	20

Провівши аналіз та визначивши середню норму рентабельності в галузі в порівняні із банківським відсотком на вкладення можна зробити висновок, що останній є меншим, тому є сенс вкладати гроші у цей проект.

За результатами аналізу таблиці 4.4 було зроблено висновок, що ринок є привабливим для входження.

Наступним кроком були визначені потенційні групи клієнтів, їх характеристики та сформовано орієнтовний перелік вимог до товару для кожної групи в таблиці 4.5.

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Формування ринку потребами	Націленість на аудиторію	Різниця, що може бути у різних груп клієнтів	Потреби клієнтів до продукції
1	2	3	4	5

Продовження таблиці 4.5

1	2	3	4	5
1	Програмне забезпечення для побудови математичної моделі контролю якості ПЗ.	Аутсорс-компанії розробники програмного забезпечення та компанії, які розроблюють власне програмне забезпечення.	Різні вимоги до підтримки програмного забезпечення та частоти випуску оновлень до нього.	Зручний інтерфейс, наявність інструкції з користування, кросплатформність.

Визначивши потенційні групи клієнтів було проведено аналіз ринкового середовища: складено таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (таблиці 4.6, 4.7).

Таблиця 4.6 – Фактори загроз

№ п/п	Фактори	Можливі загрози та їх зміст	Як компанія може відреагувати
1	Конкуренція	Вихід на ринок продуктів з кращими	Вийти на ринок акцентувавши увагу на переваги власного продукту.

Продовження таблиці 4.6

1	2	3	4
		характеристиками та умовами надання послуг.	Вдосконалення технічних моментів власного продукту. Обрати цільову аудиторію і запропонувати новий продукт по більш низькій ціні та більш якісній умові підтримці продукції.
2	Зміна потреб користувачів	Користувачу потрібне буде програмне забезпечення з іншим функціоналом.	Передбачити можливість додавання нового функціоналу у майбутньому.

Таблиця 4.7 – Фактори можливостей

№ п/п	Фактор	Можливості	Як компанія може відреагувати
1	Конкуренція	Відсутність аналогів на українському ринку для вітчизняного користувача.	Адаптація продукту до особливостей потреб на домашньому ринку.
2	Поява альтернативних методів моделювання	З’являються методи моделювання, що будуть більш легкими в освоєнні та праці з ним.	Розширення можливостей та максимальне спрощення в роботі з продуктом для користувачів.

Надалі було проведено аналіз пропозиції: визначили загальні риси конкуренції на ринку в таблиці 4.8.

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості, що зустрінуться в конкурентному середовищі	Як дана характеристика може проявитися	Як вона вплине на підприємство
1.Вказати тип конкуренції – монополія.	На ринку майже відсутні компанії-конкуренти з аналогічною продукцією.	Підтримка якості продукту, постійний розвиток, нововведення, вдосконалення, оновлення та підтримка.
2.За рівнем конкурентної боротьби - міжнародний.	Компанії-конкуренти з інших країн.	Створити основу продукту таким чином, щоб можна було легко переробити його для використання у галузях інших країн.
3.За галузевою ознакою - внутрішньогалузева	Продукт може використовуватись в одній галузі, але її різних сферах.	Постійне вдосконалення продукту, що не має прив’язки до сфери, але має до галузі
4.Конкуренція за видами товарів: - товарно-видова	Конкуренція між видами програмного продукту та їх особливостями.	Створити продукт, враховуючи недоліки конкурентів.

Продовження таблиці 4.8

5. За характером конкурентних переваг - цінова	Вдосконалення технології створення програмного продукту, щоб витрати на його створення та підтримку були нижчими.	Використання доступних та дешевих технологій для розробки в порівнянні з конкурентами, але тільки, якщо ці технології відповідають необхідним вимогам якості.
6. За інтенсивністю - не марочна	Бренд присутній, але його роль незначна	Реклама, участь у конференціях, семінарах, виставках.

Було проведено аналіз конкуренції у галузі за моделлю М. Портера (табл. 4.9).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти	Потенційні конкуренти	Постачальники	Клієнти	Аналоги
	Навести перелік прямих конкурентів	Визначити бар’єри входження в ринок	Визначити фактори сили постачальників	Визначити фактори сили споживачів	Фактори загроз з боку аналогів, що можуть замінити продукт
	SNSP	Наявність вже	-	Якість продукту	Більш

Продовження таблиці 4.9

1	2	3	4	5	6
		існуючих рішень.		та його підтримка.	авторитетний розробник, що підтримує свій продукт.
Висновки	На вітчизняному ринку, даний конкурент відсутній.	Є можливість віходу на внутрішній ринок без конкурентів.	-	Вимоги клієнтів такі, як зручний інтерфейс, простота користування, якість програмного продукту.	Випустити продукт, що буду не гірше, ніж у закордонного конкурента та розширювати функціонал.

Провівши аналіз таблиці 4.9 було зроблено висновок щодо можливості роботи на ринку з огляду на конкурентну ситуацію та щодо характеристик, які повинен мати проект, щоб бути конкуренто спроможним на ринку.

На основі аналізу конкуренції, проведеного в таблиці 4.9, а також із урахуванням характеристик ідеї проекту (таблиця 4.2), вимог споживачів до товару (таблиця 4.5) та факторів маркетингового середовища (таблиці 4.6, 4.7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлено у таблиці 4.10.

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Впливи конкурентоспроможності	Обґрунтування впливів
1	Ціна	Більш доступна ціна приваблює клієнтів.
2	Кросплатформність	Можливість використання продукту на будь-якій платформі або системі.
3	Орієнтованість на кінцевого споживача	Продукт орієнтований на взаємодію з клієнтом.

За визначеними факторами конкурентоспроможності (табл. 4.10) проведено аналіз сильних та слабких сторін стартап-проекту в таблиці 4.11.

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів конкурентів у порівнянні з… (назва підприємства)						
			3	2	1	0	1	2	3
1	Ціна	5					*		
2	Кросплатформність	0			*				
3	Орієнтованість на кінцевого споживача	7					*		

Останнім етапом ринкового аналізу можливостей впровадження проекту являється складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) в таблиці 4.12 на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (таблиця 4.11). Перелік загроз та ринкових можливостей було створено за допомогою аналізу загроз та можливостей у маркетинговому

середовищі. Ризикові загрози та ринкові можливості спричинені (очікуваними) наслідками факторів і на відміну від них, ще не реалізовані на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходу потенційних споживачів є фактором загрози, на підставі якого можна передбачити збільшення формування цінового фактору при відборі товару і, як наслідок, цінової конкуренції (і це є ринкова загроза).

Таблиця 4.12 – SWOT-аналіз стартап-проекту

Сильні сторони: Ціна, орієнтованість на кінцевого споживача, кросплатформність.	Слабкі сторони: Складність реалізувати продукцію за кордоном.
Можливості: Відсутність конкуренції на внутрішньому ринку.	Загрози: Зміна курсу потреб користувачів, при відсутності конкуренції пильна увага з боку відповідних органів влади.

Засновуючись на SWOT-аналізі буде розроблено альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час ринкової реалізації з огляду на потенційні проекти конкурентів, що також можуть бути виведені на ринок (таблиця 4.9, аналіз потенційних конкурентів). Визначені альтернативи були проаналізовані з точки зору строків та ймовірності отримання ресурсів (таблиця 4.13).

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	2	3	4
1	Безкоштовне розповсюдження демо-версії створеного ПП.	85%	18 місяців

Продовження таблиці 4.13

1	2	3	4
2	Створення ПП з подальшим розповсюдженням за певну оплату.	65%	18 місяців

Після аналізу було обрано альтернативу №1.

4.4 Розроблення ринкової стратегії ринку

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів, які приведенні в таблиці 4.14.

Таблиця 4.4. – Вибір цільових груп потенційних споживачів

№ п/п	Опис груп цільових клієнтів	Готовність скористатися продуктом	Можливий попит	Інтенсивність конкуренції	Простота входу у сегмент
1	ІТ-компанії	Готові	65%	Мінімальна	Середня
Які цільові групи обрано: ІТ-компанії.					

Згідно з аналізом потенційних груп споживачів, цільова група, для якої пропонується продукт, буде визначено стратегію входження на ринок – це диференційована маркетингова стратегія (компанія працює з декількома сегментами).

Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (таблиця 4.15).

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	Альтернативні шляхи розвитку проекту	Стратегія охоплення ринку	Конкурентоспроможні позиції	Базова стратегія розвитку*
1		Визначити потреби сучасного ринку для кожної з груп, розробити відповідно до них стратегії приваблення клієнтів та маркетингової компанії.	Цінова політика, універсальність продукту.	Стратегія диференціації.

Наступним кроком обрано стратегію конкурентної поведінки (таблиця 4.16).

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Як буде компанія шукати нових споживачів?	Чи буде компанія копіювати товари конкурентів?	Стратегія конкурентної поведінки*
1	2	3	4	5

Продовження таблиці 4.16

1	2	3	4	5
	Першопроходець»	Шукати нових	Ні	Стратегія заняття конкурентної ніші.

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту (див. таблицю 4.5), а також в залежності від обраної базової стратегії розвитку (таблиця 4.15) та стратегії конкурентної поведінки (таблиця 4.16) розроблено стратегію позиціонування (таблиця 4.17), що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17 – Визначення стратегії позиціонування

№ п/п	Які вимоги до товару в цільової аудиторії	Базова стратегія розвитку	Конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	2	3	4	5
1	Легкість в освоєнні та розумінні користування, зручний ГІК,	Стратегія диференціації	Позиція на основі порівняння фірми з	Економія часу; Зручність застосування;

Продовження таблиці 4.17

1	2	3	4	5
1	надійний, швидкий, точний та достовірний ПП для побудови математичної моделі контролю якості ПЗ	товарами конкурентів; Відмінні особливості споживача.		Практичність.

Отриманий результат дав узгоджену систему рішень щодо ринкової поведінки стартап-компанії, яка визначає напрями роботи стартап-компанії.

4.5 Розроблення маркетингової програми стартап-проекту

Розроблено маркетингову концепцію продукту, який отримує споживач. Таблиця 4.18 підсумовує результати попереднього аналізу конкурентоспроможності продукту. Концепція продукту – це письмовий опис фізичних та інших характеристик сприйманих споживачем продуктів і всіх переваг, які вона обіцяє певній групі споживачів.

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреби	Вигоди, які товар надає аудиторії	Ключові переваги перед конкурентами
1	2	3	4

Продовження таблиці 4.18

1	2	3	4
1	Швидкість отримання результату	Проста побудова моделі, з якої користувач отримує потрібні йому дані.	Відсутність необхідності звертатися до компанії аутсорсеру для оптимізації роботи.
2	Зручність застосування	Існує інструкція за допомогою якої користувач легко оволодіє користуванням ПП.	Використано математичний підхід для аналізу моделі розробки ПЗ або аудиту його якості.

Розроблено трирівневу маркетингову модель для продукту: з'ясовано ідею продукту та / або послуги, його фізичні компоненти та характеристики процесу підготовки в таблиці 4.19.

Рівень 1. При розробці ідеї продукту, питання полягає в тому, чим є даний продукт, його основна користь, спосіб вирішення потреб та / або проблем. Ця проблема безпосередньо пов'язана зі створенням технічних специфікацій при створенні конструкторської документації продукту.

Рівень 2. Цей рівень представляє собою рішення щодо того, як продукт дійсно реалізується / включає якість, властивості, дизайн, упаковку, ціну.

Рівень 3. Товар з підкріпленням (технічне обслуговування) - додаткові послуги та переваги для споживача на основі продукту відповідно до плану та продукту з точки зору фактичної ефективності (забезпечення якості, доставки, термінів оплати тощо).

Таблиця 4.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
1	2

Продовження таблиці 4.19

1	2		
І.Товар за задумом	ПП, який дозволяє знаходити ймовірні помилки та слабкі місця програмних проектів на початкових стадіях їх життєвого циклу.		
ІІ.Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Кросплатформність. 2.Користувачу не знадобиться багато часу для ознаймлення з тим, я треба працювати в продуктом. 3.Не потребує великих обчислювальних ресурсів.		
	Якість: стандарти (ДСТУ, ISO), норми, параметри тестування тощо		
	Марка: назва організації-розробника + назва товару		
Потенційний товар буде захищено від копіювання: патентування, сертифікати відповідності.			

Після створення маркетингової моделі для продукту треба звернути увагу, що проект захищений від копіювання за допомогою ноу-хау. Наступним кроком є встановлення цінових меж, які необхідно враховувати при визначенні ціни потенційного продукту (остаточне визначення ціни буде зроблено в ході фінансово-економічного аналізу проекту), що має на меті визначення ціни, аналіз цін аналогічних товарів або замінників і аналіз рівня доходів цільової групи споживачів, яке відображено в таблиці 4.20. Аналіз проводився експертним методом.

Таблиця 4.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	10000$	100000$	9000-12000	2000-2500$

Наступним кроком буде визначатися оптимальна системи збуту, в межах якої буде прийняте відповідне рішення у таблиці 4.21. А останньою складовою маркетингової програми є розробка концепції маркетингової комунікації, яка базується на попередньо відібраній позиційній базі, конкретній специфіці поведінки замовника у таблиці 4.22.

Таблиця 4.21 – Формування системи збуту

№ п/п	Закупельна поведінка потенційних клієнтів	Методи збуту	Канали збуту	Системи збуту
1	2	3	4	5
1	Цільові клієнти – компанії, які бажають впровадити у своїй роботі сучасні засоби, які допоможуть оптимізувати процес розробки та знаходити ймовірні	Встановлення контактів зі споживачами і підтримання їх. Формування попиту і стимулювання збуту. Дослідницька	Один (від виробника одразу споживачу).	Прямий канал збуту до споживача, мінімізувати збутові витрати, розвиток маркетингового спілкування зі споживачем. Випускати товар

Продовження таблиці 4.21

1	2	3	4	5
	слабкі місця в моделі розробки або контролю якості ПЗ ще на початкових стадіях робіт.	робота зі збору маркетингової інформації. Доробка товару, виходячи з потреб конкретного споживача.		в цифровому вигляді.

Таблиця 4.22 – Концепція маркетингових комунікацій

№ п/п	Закупельна поведінка потенційних клієнтів	Канали комунікацій з потенційними клієнтами	Ключові позиції для позиціонування товару	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Бажання клієнтів зекономити час при розробці власного програмного забезпечення	Конференції, виставки, реклама в інтернеті на тематичних сайтах.	Виставки присвячені розробці програмного забезпечення та реклама в інтернеті	Донести основну ідею, цінову політику та можливості співпраці.	Продемонструвати нове рішення та новий погляд на існуюче питання контролю якості ПЗ.

Результатом підрозділу стала ринкова (маркетингова) програма, що включає концепції продуктів, продажу, стимулювання збуту і попередній аналіз цін, заснований на потребах потенційних клієнтів, конкурентні переваги ринку, ідея, ситуація і його динаміка, на якому буде реалізовано проект.

4.6 Висновки

В даному розділі було проведено аналіз програмного продукту у якості стартап проекту. Можна зазначити, що у проекті є можливість комерціалізації, оскільки ринок потребує якісний продукт, що надає можливість створювати моделі нелінійних-нестаціонарних процесів.

На ринку наявна монополістична конкуренція, існує декілька фірм-конкурентів, але їх товар дещо відрізняється, тому вихід на ринок не буде легким і потребує грамотної стратегії виходу. Для впровадження ринкової реалізації проекту слід обрати альтернативу, яка передбачає розробку програмного продукту з подальшим розповсюдженням за певну плату.

Можна сказати, що подальший розвиток проекту є доцільним, оскільки він знайде свою цільову аудиторію.

ВИСНОВКИ

В процесі виконання дипломної роботи було розроблено програмне забезпечення, в основу якого була закладена математична модель процесу контролю якості ПЗ. За допомогою якого можна знаходити помилки в організації розробки ПЗ на будь-якій стадії її розробки, а також визначати час, необхідний на верифікацію цього програмного продукту, таким чином виявивши слабкі місця в модельованому процесі та ліквідувавши їх оптимізувати працю розробників та інженерів із забезпечення якості ПЗ.

Метою даної роботи було створення ПЗ, за допомогою якого можна створити математичну модель процесу контролю якості програмного забезпечення та розробити автоматизовану систему пошуку помилок та слабких місць проектів на будь-яких стадіях їх життєвого циклу у вигляді ПП. В якості мови моделювання використовуються мережі Петрі, а їх аналіз допомагає знайти помилки в моделі. Для досягнення поставленої мети були вирішені наступні задачі:

- провели огляд основного поняття контролю якості програмного забезпечення;
- проаналізували математичне моделювання з використанням мереж Петрі;
- створено та проаналізовано модель аудиту якості програмного забезпечення шляхом та виявлені недоліки в ній;
- розроблено проект програмного забезпечення контролю якості, який включає в себе три рівні представлення: ескізний проект; технічний проект та робочий, застосовуючи при цьому всі існуючі аспекти для розуміння проекту, а саме структуру програми; керування; стани програми при виконанні;

В результаті виконання магістерської роботи було розроблено ПЗ по контролю якості з допомогою мереж Петрі, яке дозволяє досягти наступних

результатів, що наведені нижче.

1. Виявляти помилки в логіці вихідної моделі процесів, які можуть вести до багаторазової ініціації однієї і тієї ж діяльності в один момент часу, що, у свою чергу, веде до збільшення витрат ресурсів, збільшення проектного часу і порушення логіки моделі процесу.

2. Перевірити, що всі робочі етапи розробки ПЗ виконуються лише один раз і присутні в кількості одного примірника на всьому протязі виконання тієї або іншої дії. Це дозволить гарантувати відсутність дублювання проектної інформації, а в разі необхідності усунути зайві витрати, необхідні для цього.

3. Виявити можливі тупикові ситуації і усунути їх. Тупик в мережі Петрі – це перехід (або множина переходів), який не може бути запущений. Виконання моделі процесу не має блокуватися без можливості подальшого продовження. Модель процесу повинна бути вільна від наявності тупикових ситуацій.

4. Оптимізувати моделі процесу розробки програмного забезпечення, цей процес проводиться поетапно рішенням наступних підзадач: збільшенням рівня паралелізму, пошуком пасивних позицій і переходів та їх виключенням.

5. Важливою якісною характеристикою створеної математичної моделі є рівномірність розподілу завантаження ресурсів різних процесів розробки ПП. Аналіз даної характеристики дозволяє виявити можливі простої ресурсів при виконанні проекту, визначити мінімальну, максимальну, середню та інші кількісні характеристики завантаження ресурсів.

Працездатність нової програми була перевірена на прикладі створеній моделі аудиту якості ПЗ. Модель використовується для визначення проблем та невідповідностей в досліджуваному процесі.

З урахуванням усього вищенаведеного можна сказати, що мета, сформована, на початку магістерської роботи, була повністю досягнута.

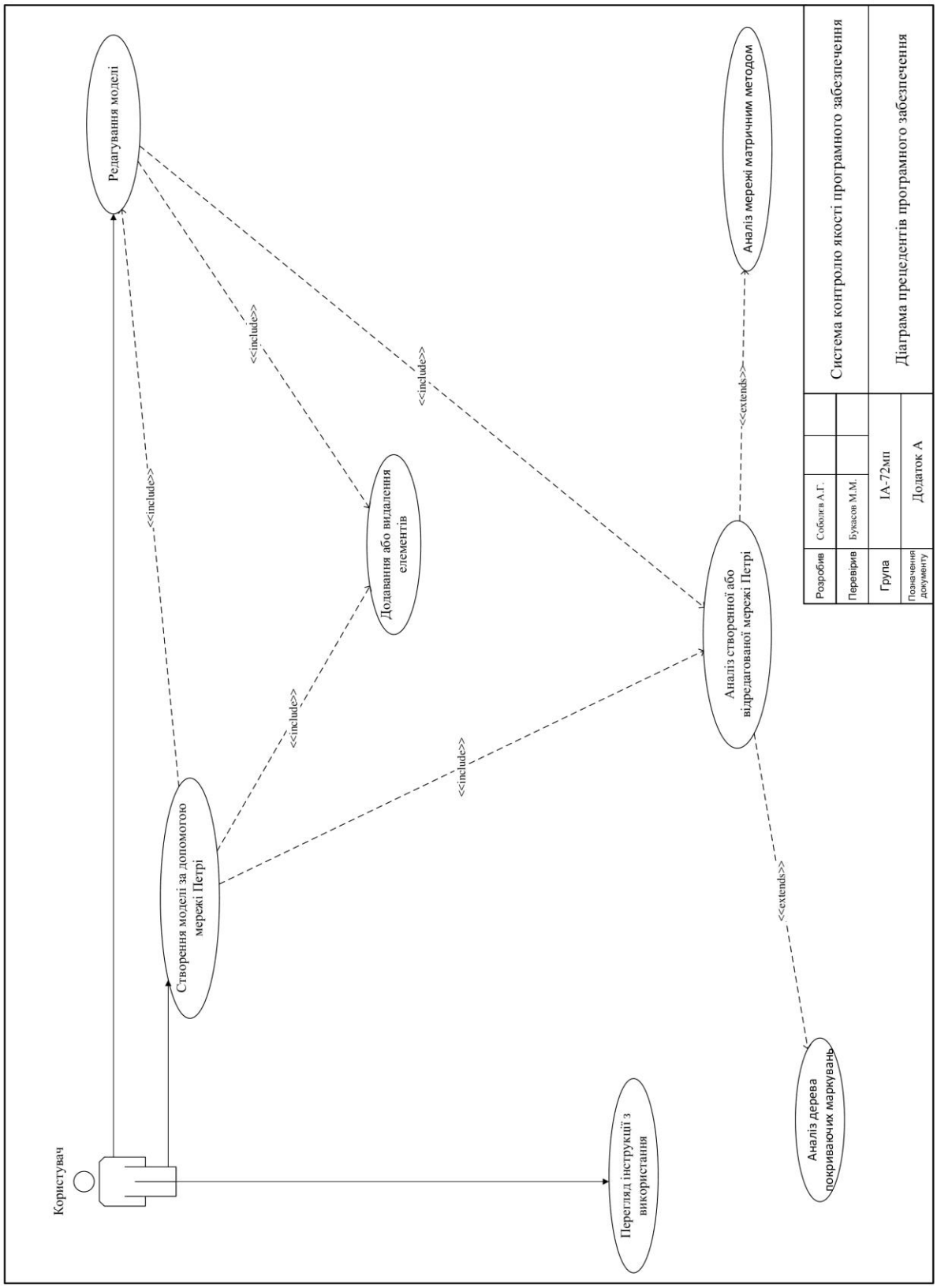
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

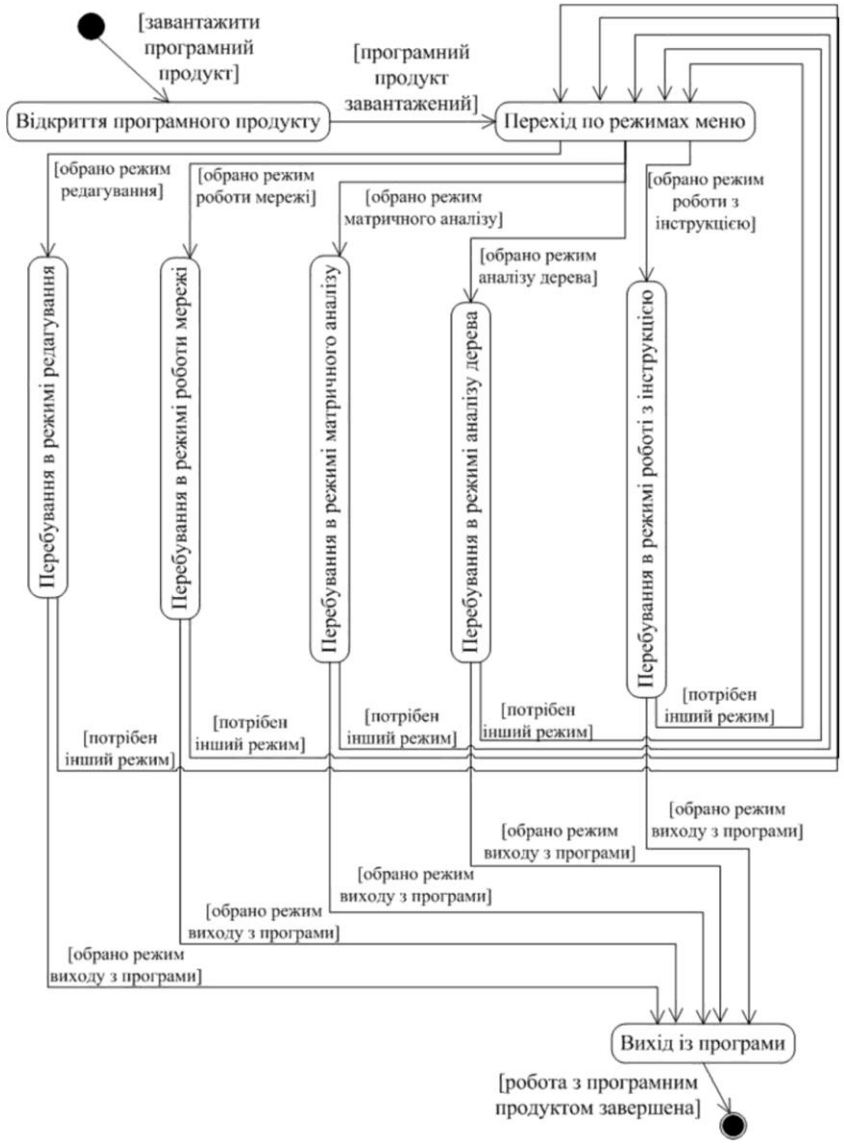
1. Матвеева Л. Є. Инженерия качества процессов производства программных систем с помощью сетей Петри / Л. Є. Матвеева - Киев – 277 - 283 с.
2. Матвеева Л. Є. Процес розробки програмного забезпечення. Від теорії до практики / Л. Є.Матвеева, В. А. Волков– Київ. – 2008. – 402 с.
3. Котов, В. Е. Сети Петри / В. Е. Котов. – М.:Наука, 1984. – 157 с.
4. Никулина, Н. О. Применение аппарата сетей Петри для моделирования экономических процессов: методические указания к лабораторным работам по курсу “ЛВС и распределенная обработка данных в банках для подготовки инженеров по специальности “Прикладная информатика в экономике” / Н. О. Никулина, Е.Б. Старцева. –Уфа: Уфимск. гос. авиац. техн. ун-т. – 2001. – 32 с.
5. Хачатуров, А. Р. Моделирование процессов разработки программного обеспечения учебного назначения при помощи сетей Петри / А. Р. Хачатуров, И. Ф. Бабалова // Научная сессия МИФИ-2007. Сб. научных трудов Т.12. М.: МИФИ, 2007, с. 159-160.
6. Питерсон Дж. Теория сетей Петри и моделирования систем. - М: Мир. 1984. -264 с.
7. Лескин А.А., Мальцев П.А., Спиридонов А.М. Сети Петри в моделировании и управлении. Л.: Наука, 1989. 133с.
8. Якість програмного забезпечення [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Якість_програмного_забезпечення.
9. Издана по-русски в сборнике Ф. Брукс. Мифический человеко-месяц, или Как создаются программные системы. СПб.: Символ-Плюс, 1999
10. Boehm B. W. Software Engineering Economics. / B. W. Boehm - Englewood Cliffs - NJ: Prentice-Hall PTR - 1981. Русский перевод: Боэм

Б. У. Инженерное проектирование программного обеспечения. / Б. У . Боэм - М.: Радо и связь, 1985.

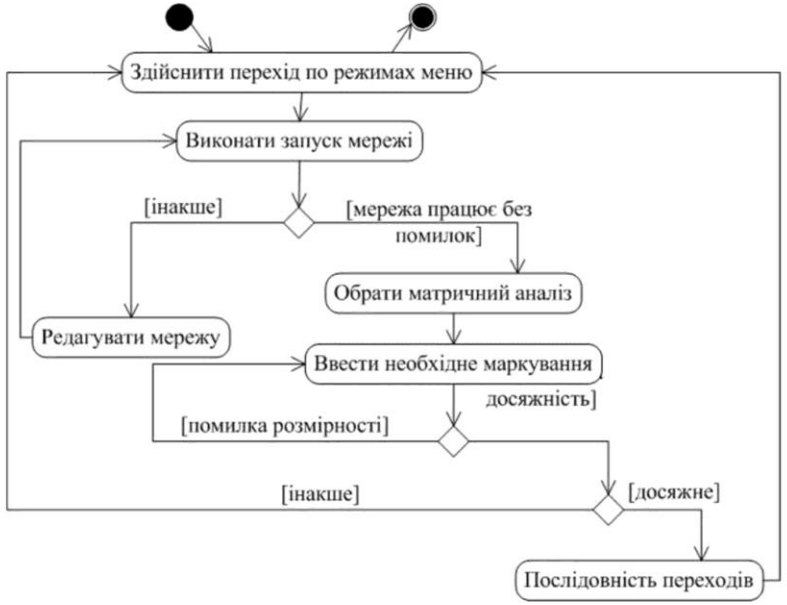
11. Оннгденко Б.О.. Супруненко О.О. Управляючі мережі Петрі, як засіб моделювання та автоматизованого аналізу алгоритмічних конструкцій. // Вісник запорізького національного \ніверситет\\ -2009.-.N21.-С. 197-203.
12. Васильев В.В.. Кузьмук В.В. Сети Петри, параллельные алгоритмы и модели мультипроцессорных систем - К.: Наукова думка. 1990 - 216 с.
13. Miller H. J. Sanders. Scoping the Global Market: Size Is Just Part of the Story. IT Professional, 1(2) / H. J. Miller - 49-54 - 1999.
14. Karp R., Miller R., Parallel Program Schemata, RC-2053, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1968, pp. 54
15. Чим тестування чорного ящика відрізняється від білого [Електронний ресурс] – Режим доступу до ресурсу: http://moyaosvita.com.ua/tehnologii/chim-testuvannya-chornogo-yashhika-vidriznyayetsya-vid-bilogo/.
16. Техніки тестування — шпаргалка для QA [Електронний ресурс] – Режим доступу до ресурсу: https://www.quality-assurance-group.com/tehniky-testuvannya-shpargalka-dlya-qa-chastyna-2/.

Додаток А. Діаграма класів

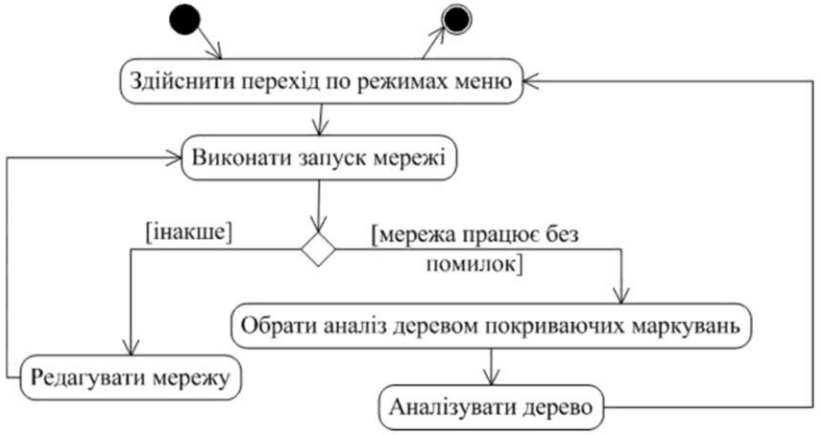




Розробив	Соболєв А.Г.			Система контролю якості програмного забезпечення
Перевірив	Букасов М.М.			
Група	ІА-72мп			Діаграма станів програмного забезпечення
Позначення документу	Додаток А			



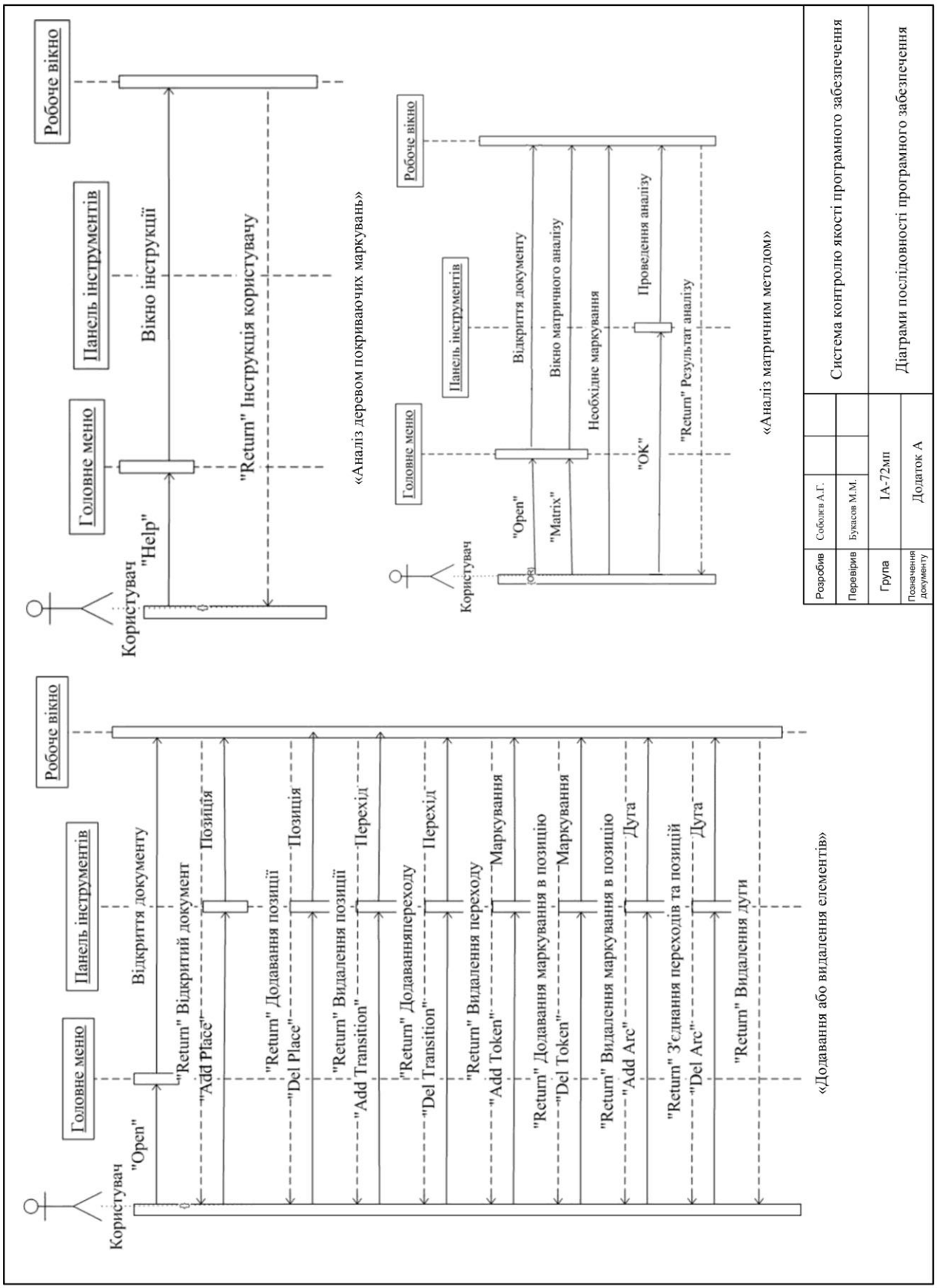
«Аналіз мережі матричним способом»

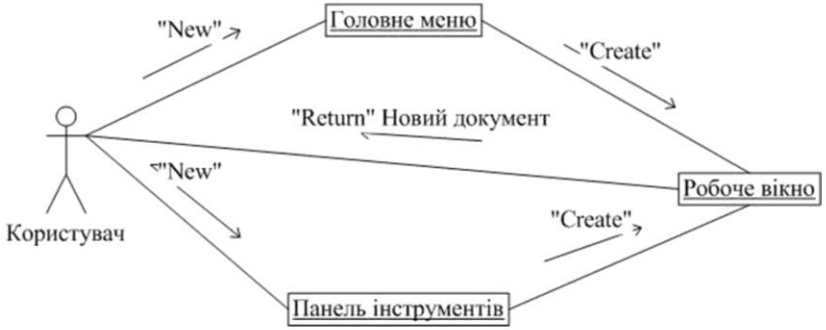


«Аналіз мережі деревом покриваючих маркувань»

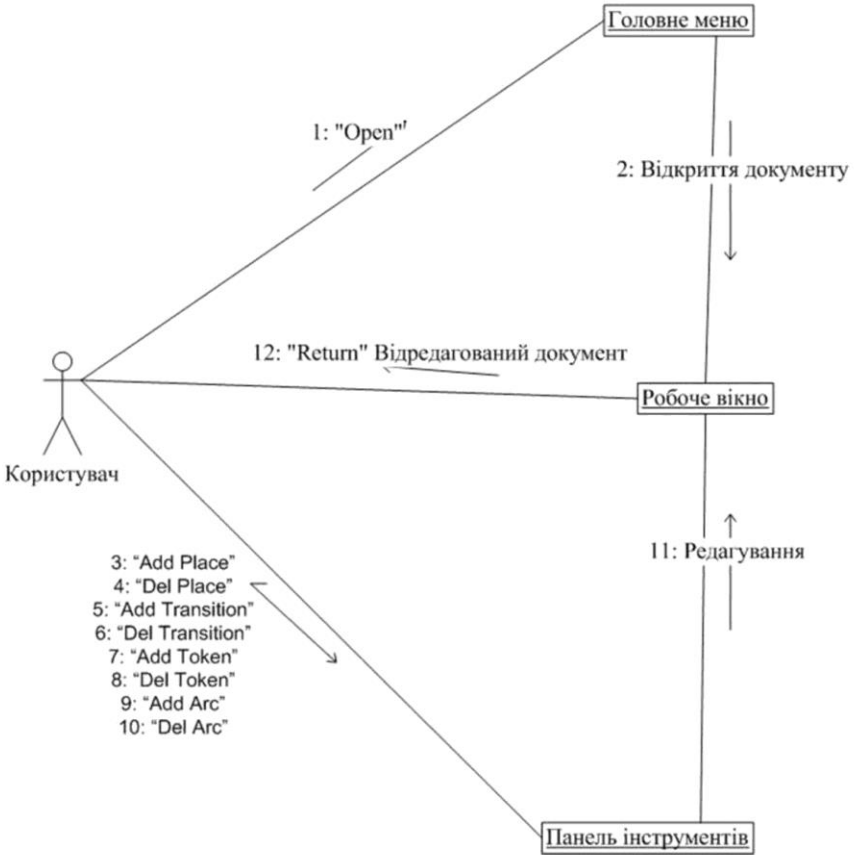
«Перевірка працездатності мережі»

Розробив	Соболєв А.Г.			Система контролю якості програмного забезпечення
Перевірив	Букасов М.М.			
Група	ІА-72мп			Діаграми діяльності програмного забезпечення
Позначення документу	Додаток А			


Користувач
Головне меню
Панель інструментів
Робоче вікно
"Open"
"Return" Відкритий документ
Відкриття документу
"Add Place"
Позиція
"Return" Додавання позиції
"Del Place"
"Return" Видалення позиції
"Add Transition"
Перехід
"Return" Додаванняпереходу
"Del Transition"
"Return" Видалення переходу
"Add Token"
Маркування
"Return" Додавання маркування в позицію
"Del Token"
"Return" Видалення маркування в позицію
"Add Arc"
Дуга
"Return" З'єднання переходів та позицій
"Del Arc"
"Return" Видалення дуги
«Додавання або видалення елементів»
"Help"
Вікно інструкції
"Return" Інструкція користувачу
«Аналіз деревом покриваючих маркувань»
"Open"
"Matrix"
Відкриття документу
Вікно матричного аналізу
Необхідне маркування
"OK"
Проведення аналізу
"Return" Результат аналізу
«Аналіз матричним методом»
Розробив
Соболєв А.Г.
Перевірив
Бусасов М.М.
Група
ІА-72мп
Позначення документу
Додаток А
Система контролю якості програмного забезпечення
Діаграми послідовності програмного забезпечення



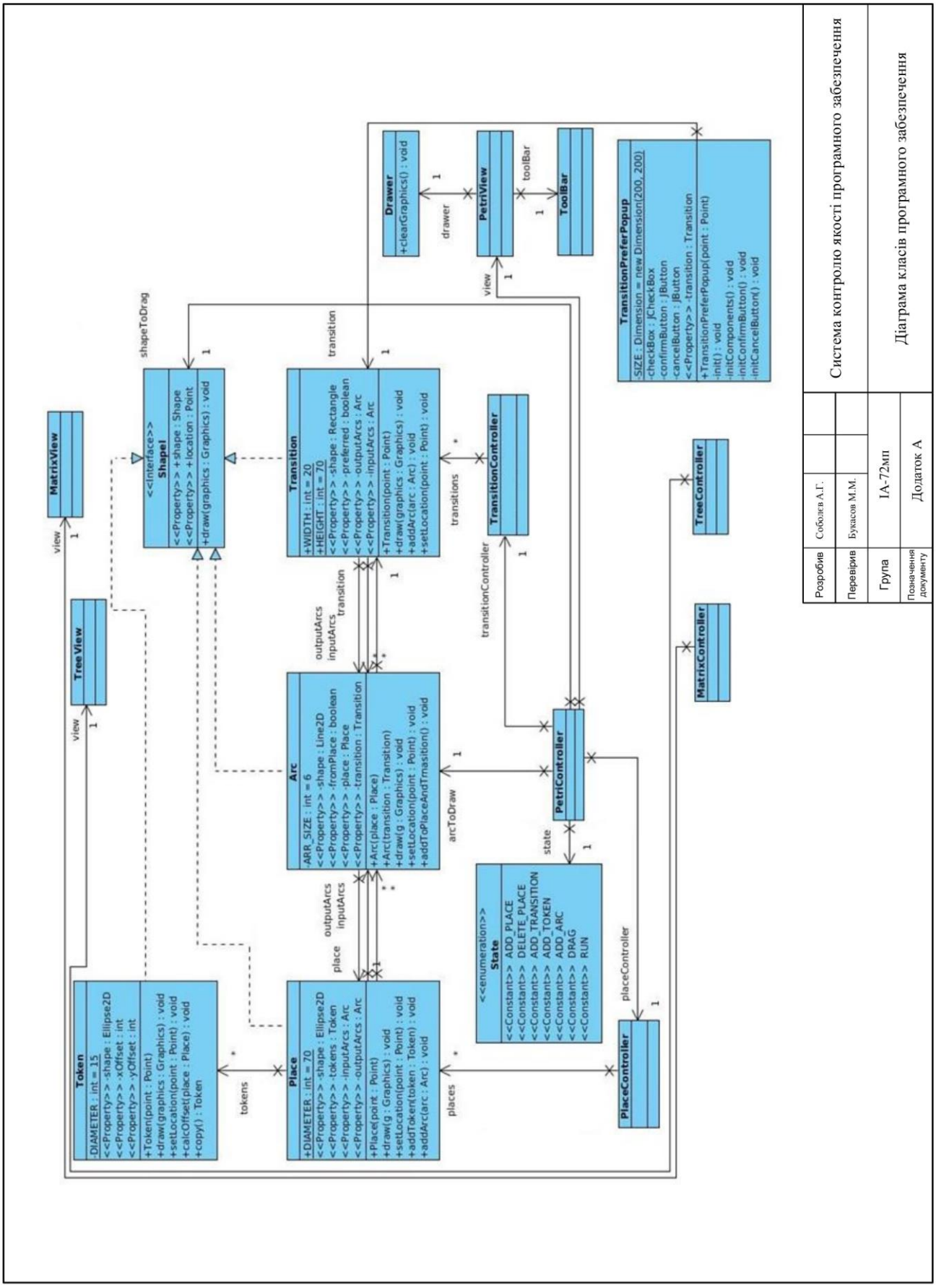
«Створення нового документу»



«Додавання або видалення елементів»

«Редагування документу»

Розробив	Соболєв А.Г.			Система контролю якості програмного забезпечення
Перевірив	Букасов М.М.			
Група	ІА-72мп			Діаграми кооперацій програмного забезпечення
Позначення документу	Додаток А			


Token
-DIAMETER : int = 15
<<Property>> -shape : Ellipse2D
<<Property>> -xOffset : int
<<Property>> -yOffset : int
+Token(point : Point)
+draw(graphics : Graphics) : void
+setLocation(point : Point) : void
+calcOffset(place : Place) : void
+copy() : Token
Place
+DIAMETER : int = 70
<<Property>> -shape : Ellipse2D
<<Property>> -tokens : Token
<<Property>> -inputArcs : Arc
<<Property>> -outputArcs : Arc
+Place(point : Point)
+draw(g : Graphics) : void
+setLocation(point : Point) : void
+addToken(token : Token) : void
+addArc(arc : Arc) : void
Arc
-ARR_SIZE : int = 6
<<Property>> -shape : Line2D
<<Property>> -fromPlace : boolean
<<Property>> -place : Place
<<Property>> -transition : Transition
+Arc(place : Place)
+Arc(transition : Transition)
+draw(g : Graphics) : void
+setLocation(point : Point) : void
+addToPlaceAndTmasition() : void
Transition
+WIDTH : int = 20
+HEIGHT : int = 70
<<Property>> -shape : Rectangle
<<Property>> -preferred : boolean
<<Property>> -outputArcs : Arc
<<Property>> -inputArcs : Arc
+Transition(point : Point)
+draw(graphics : Graphics) : void
+addArc(arc : Arc) : void
+setLocation(point : Point) : void
<<interface>>
Shape
<<Property>> +shape : Shape
<<Property>> +location : Point
+draw(graphics : Graphics) : void
<<enumeration>>
State
<<Constant>> ADD_PLACE
<<Constant>> DELETE_PLACE
<<Constant>> ADD_TRANSITION
<<Constant>> ADD_TOKEN
<<Constant>> ADD_ARC
<<Constant>> DRAG
<<Constant>> RUN
Drawer
+clearGraphics() : void
TransitionPreferPopup
-SIZE : Dimension = new Dimension(200, 200)
-checkBox : JCheckBox
-confirmButton : JButton
-cancelButton : JButton
<<Property>> -transition : Transition
+TransitionPreferPopup(point : Point)
-init() : void
-initComponents() : void
-initConfirmButton() : void
-initCancelButton() : void
MatrixView
TreeView
PetriView
ToolBar
TransitionController
TreeController
MatrixController
PetriController
PlaceController
view
shapeToDrag
transition
drawer
toolBar
tokens
places
outputArcs
inputArcs
transitions
transitionController
arcToDraw
state
placeController
Розробив
Соболєв А.Г.
Перевірив
Букасов М.М.
Група
ІА-72мп
Позначення документу
Додаток А
Система контролю якості програмного забезпечення
Діаграма класів програмного забезпечення

Додаток Б. Лістинг Коду

Лістинг ToolBar

```
package com.petri.view;

import java.awt.FlowLayout;
import javax.swing.*;

/**
 * @author soboliev
 */
public class ToolBar extends JToolBar {

    private JButton newButton;
    private JButton runButton;
    private JButton stopButton;
    private ButtonGroup buttonGroup;
    private JToggleButton dragButton;
    private JToggleButton deleteTransitionButton;
    private JToggleButton addTransitionButton;
    private JToggleButton deleteTokenButton;
    private JToggleButton addTokenButton;
    private JToggleButton deleteArcButton;
    private JToggleButton addArcButton;
    private JToggleButton deletePlaceButton;
    private JToggleButton addPlaceButton;

    public ToolBar() {
        init();
    }

    private void init() {
        setLayout(new FlowLayout(FlowLayout.LEFT));
        setRollover(true);
        buttonGroup = new ButtonGroup();
        initNewFileButton();
        initRunButton();
        initStopButton();
        initDragButton();
        initDeleteTransitionButton();
        initAddTransitionButton();
        initDeleteTokenButton();
        initAddTokenButton();
        initDeleteArcButton();
        initAddArcButton();
        initDeletePlaceButton();
        initAddPlaceButton();
    }

    private void initNewFileButton() {
        newButton = new JButton();
        newButton.setIcon(new
ImageIcon(ToolBar.class.getClassLoader().getResource("newrks.png")));
        newButton.setText("New");

newButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

newButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
        buttonGroup.add(newButton);
        add(newButton);
```

```
        }

        private void initRunButton() {
            runButton = new JButton();
            runButton.setIcon(new
ImageIcon(ToolBar.class.getClassLoader().getResource("Starts.png"))); //
NOI18N
            runButton.setText("Run");

runButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

runButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
            buttonGroup.add(runButton);
            add(runButton);
        }

        private void initStopButton() {
            stopButton = new JButton();
            stopButton.setIcon(new
ImageIcon(ToolBar.class.getClassLoader().getResource("Stop.png"))); // NOI18N
            stopButton.setText("Stop");
            stopButton.setFocusable(false);

stopButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

stopButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
            buttonGroup.add(stopButton);
            add(stopButton);
        }

        private void initDragButton() {
            dragButton = new JToggleButton();
            dragButton.setIcon(new
javax.swing.ImageIcon(ToolBar.class.getClassLoader().getResource("moveeworks.
png"))); // NOI18N
            dragButton.setText("Drag Element");
            dragButton.setFocusable(false);

dragButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

dragButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
            buttonGroup.add(dragButton);
            add(dragButton);
        }

        private void initDeleteTransitionButton() {
            deleteTransitionButton = new JToggleButton();
            deleteTransitionButton.setIcon(new
javax.swing.ImageIcon(ToolBar.class.getClassLoader().getResource("transitione
rases.png"))); // NOI18N
            deleteTransitionButton.setText("Del Transition");
            deleteTransitionButton.setFocusable(false);

deleteTransitionButton.setHorizontalTextPosition(javax.swing.SwingConstants.C
ENTER);

deleteTransitionButton.setVerticalTextPosition(javax.swing.SwingConstants.BOT
TOM);
            buttonGroup.add(deleteTransitionButton);
            add(deleteTransitionButton);
        }
```

```
          private void initAddTransitionButton() {
              addTransitionButton = new JToggleButton();
              addTransitionButton.setIcon(new
javax.swing.ImageIcon(ToolBar.class.getClassLoader().getResource("transition.
png"))); // NOI18N
              addTransitionButton.setText("Add Transition");
              addTransitionButton.setFocusable(false);

addTransitionButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENT
ER);

addTransitionButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM
);
              buttonGroup.add(addTransitionButton);
              add(addTransitionButton);
          }

          private void initDeleteTokenButton() {
              deleteTokenButton = new JToggleButton();
              deleteTokenButton.setIcon(new
javax.swing.ImageIcon(ToolBar.class.getClassLoader().getResource("erasepe_arc
_up_gestureworks.png"))); // NOI18N
              deleteTokenButton.setText("Del Token");
              deleteTokenButton.setFocusable(false);

deleteTokenButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER
);

deleteTokenButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
              buttonGroup.add(deleteTokenButton);
              add(deleteTokenButton);
          }

          private void initAddTokenButton() {
              addTokenButton = new JToggleButton();
              addTokenButton.setIcon(new
javax.swing.ImageIcon(ToolBar.class.getClassLoader().getResource("stroke_shap
e_arc_up_gestureworks.png"))); // NOI18N
              addTokenButton.setText("Add Token");
              addTokenButton.setFocusable(false);

addTokenButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

addTokenButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
              buttonGroup.add(addTokenButton);
              add(addTokenButton);
          }

          private void initDeleteArcButton() {
              deleteArcButton = new JToggleButton();
              deleteArcButton.setIcon(new
javax.swing.ImageIcon(ToolBar.class.getClassLoader().getResource("arcs.png"))
); // NOI18N
              deleteArcButton.setText("Del Arc");
              deleteArcButton.setFocusable(false);

deleteArcButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

deleteArcButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
              buttonGroup.add(deleteArcButton);
              add(deleteArcButton);
          }
```

```
        private void initAddArcButton() {
            addArcButton = new JToggleButton();
            addArcButton.setIcon(new
javax.swing.ImageIcon(ToolBar.class.getClassLoader().getResource("arcadds.png
")));  // NOI18N
            addArcButton.setText("Add Arc");
            addArcButton.setFocusable(false);

addArcButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

addArcButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
            buttonGroup.add(addArcButton);
            add(addArcButton);
        }

        private void initDeletePlaceButton() {
            deletePlaceButton = new JToggleButton();
            deletePlaceButton.setIcon(new
javax.swing.ImageIcon(ToolBar.class.getClassLoader().getResource("2s.png")));
// NOI18N
            deletePlaceButton.setText("Del Place");
            deletePlaceButton.setFocusable(false);

deletePlaceButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER
);
            deletePlaceButton.setName(""); // NOI18N

deletePlaceButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
            buttonGroup.add(deletePlaceButton);
            add(deletePlaceButton);
        }

        private void initAddPlaceButton() {
            addPlaceButton = new JToggleButton();
            addPlaceButton.setIcon(new
javax.swing.ImageIcon(ToolBar.class.getClassLoader().getResource("3s.png")));
// NOI18N
            addPlaceButton.setText("Add Place");
            addPlaceButton.setFocusable(false);

addPlaceButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

addPlaceButton.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
            buttonGroup.add(addPlaceButton);
            add(addPlaceButton);
        }

        public JButton getNewButton() {
            return newButton;
        }

        public JToggleButton getAddPlaceButton() {
            return addPlaceButton;
        }

        public JToggleButton getDeletePlaceButton() {
            return deletePlaceButton;
        }

        public JToggleButton getAddTransitionButton() {
            return addTransitionButton;
```

```
    }

    public JToggleButton getDragButton() {
        return dragButton;
    }

    public JToggleButton getAddTokenButton() {
        return addTokenButton;
    }

    public JToggleButton getAddArcButton() {
        return addArcButton;
    }

    public JButton getRunButton() {
        return runButton;
    }
}
```

Лістинг PetriView

```
package com.petri.view;
import java.awt.BorderLayout;
import java.awt.Dimension;
import javax.swing.*;

/**
 * @author soboliev
 */
public class PetriView extends JFrame {

    private JMenuBar menuBar;
    private JMenu fileMenu;
    private JMenu analysisMenu;
    private JMenuItem newFileMenuItem;
    private JMenuItem exitMenuItem;
    private JMenuItem matrixMenu;
    private ToolBar toolBar;
    private JPanel bottomPanel;
    private Drawer drawer;

    public PetriView() {
        initFrame();
        initComponents();
        setLocationRelativeTo(null);
    }
    private void initFrame() {
        setMinimumSize(new Dimension(1170, 700));
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLayout(new BorderLayout());
    }

    private void initComponents() {
        initBottompanel();
        initDrawer();
    }

    private void initBottompanel() {
        bottomPanel = new JPanel();
        bottomPanel.setLayout(new BorderLayout());
        initMenu();
        initToolbar();
```

```
            add(bottomPanel, BorderLayout.NORTH);
        }

        private void initMenu() {
            menuBar = new JMenuBar();
            initFileMenu();
            initAnalysisMenu();
            bottomPanel.add(menuBar, BorderLayout.NORTH);
        }

        private void initFileMenu() {
            fileMenu = new JMenu("File");
            initFileSubMenus();
            menuBar.add(fileMenu);
        }

        private void initFileSubMenus() {
            newFileMenuItem = new JMenuItem("New");
            exitMenuItem = new JMenuItem("Exit");
            fileMenu.add(newFileMenuItem);
            fileMenu.add(exitMenuItem);
        }

        private void initAnalysisMenu() {
            analysisMenu = new JMenu("Analysis");
            matrixMenu = new JMenuItem("Matrix");
            analysisMenu.add(matrixMenu);
            menuBar.add(analysisMenu);
        }

        private void initToolbar() {
            toolBar = new ToolBar();
            toolBar.setMinimumSize(new Dimension((int)
getBounds().getSize().getWidth(), 70));
            bottomPanel.add(toolBar, BorderLayout.CENTER);
        }

        private void initDrawer() {
            drawer = new Drawer();
            add(drawer, BorderLayout.CENTER);
        }

        public ToolBar getToolBar() {
            return toolBar;
        }

        public Drawer getDrawer() {
            return drawer;
        }

        public JMenuItem getMatrixMenu() {
            return matrixMenu;
        }
```

Лістинг MatrixView

```
package com.petri.view;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.text.NumberFormat;
import java.text.ParseException;
```

```
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;
import javax.swing.text.*;

/**
 * @author soboliev
 */
public class MatrixView extends JDialog {

    private static final Dimension SIZE = new Dimension(200, 90);
    private JTextField textField;
    private JButton confirmButton;
    private JButton cancelButton;

    public MatrixView() {
        initDialog();
        initComponents();
    }

    private void initDialog() {
        setTitle("Enter marking");
        setModal(true);
        setMinimumSize(SIZE);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setLayout(new BorderLayout());
        setLocationRelativeTo(null);
    }

    private void initComponents() {
        textField = new JTextField();
        textField.setPreferredSize(new Dimension(100, 35));
        confirmButton = new JButton("OK");
        cancelButton = new JButton("Cancel");
        add(textField, BorderLayout.PAGE_START);
        add(confirmButton, BorderLayout.LINE_START);
        add(cancelButton, BorderLayout.LINE_END);
    }

    public JButton getCancelButton() {
        return cancelButton;
    }

    public JButton getConfirmButton() {
        return confirmButton;
    }

    public JTextField getTextField() {
        return textField;
    }
}
```

Лістинг Arc

```
package com.petri.shape;

import java.awt.*;
import java.awt.geom.*;

/**
 * @author soboliev
 */
```

```
      public class Arc implements ShapeI {

          private final int ARR_SIZE = 6;
          private Line2D shape;
          private boolean fromPlace;
          private Place place;
          private Transition transition;

          public Arc(Place place) {
              shape = new Line2D.Double(place.getShape().getCenterX(),
place.getShape().getCenterY(),
                      place.getShape().getCenterX(),
place.getShape().getCenterY());
              this.place = place;
              fromPlace = true;
          }

          public Arc(Transition transition) {
              shape = new Line2D.Double(transition.getShape().getCenterX(),
transition.getShape().getCenterY(),
                      transition.getShape().getCenterX(),
transition.getShape().getCenterY());
              this.transition = transition;
              fromPlace = false;
          }

          @Override
          public void draw(Graphics g) {
              if (g != null) {
                  g.setColor(Color.BLACK);
                  Graphics2D graphics2D = (Graphics2D) g;
                  graphics2D.draw(shape);
                  double dx = shape.getX2() - shape.getX1(), dy =
shape.getY2() - shape.getY1();
                  double angle = Math.atan2(dy, dx);
                  int len = (int) Math.sqrt(dx * dx + dy * dy);
                  AffineTransform at =
AffineTransform.getTranslateInstance(shape.getX1(), shape.getY1());
                  at.concatenate(AffineTransform.getRotateInstance(angle));
                  Shape arrow = at.createTransformedShape(new Polygon(new
int[]{len, len - ARR_SIZE, len - ARR_SIZE, len},
                          new int[]{0, -ARR_SIZE, ARR_SIZE, 0}, 4));
                  graphics2D.fill(arrow);
              }
          }

          @Override
          public Line2D getShape() {
              return shape;
          }

          @Override
          public void setLocation(Point point) {
              shape.setLine(shape.getP1(), point);
          }

          public Place getPlace() {
              return place;
          }

          public Transition getTransition() {
              return transition;
```

```
        }

        public void setPlace(Place place) {
            this.place = place;
            if (fromPlace) {
                shape.setLine(new
Point2D.Double(place.getShape().getCenterX(), place.getShape().getCenterY()),
shape.getP2());
            } else {
                shape.setLine(shape.getP1(), new
Point2D.Double(place.getShape().getCenterX(),
place.getShape().getCenterY()));
            }
        }

        public void setTransition(Transition transition) {
            this.transition = transition;
            if (fromPlace) {
                shape.setLine(shape.getP1(), new
Point2D.Double(transition.getShape().getCenterX(),
transition.getShape().getCenterY()));
            } else {
                shape.setLine(new
Point2D.Double(transition.getShape().getCenterX(),
transition.getShape().getCenterY()), shape.getP2());
            }
        }

        public void addToPlaceAndTrnasition() {
            place.addArc(this);
            transition.addArc(this);
        }

        public boolean isFromPlace() {
            return fromPlace;
        }
    }
```

Лістинг Place

```
package com.petri.shape;

import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author soboliev

 */
public class Place implements ShapeI {

    public static int DIAMETER = 70;
    private Ellipse2D shape;
    private List<Token> tokens;
    private List<Arc> inputArcs;
    private List<Arc> outputArcs;

    public Place(Point point) {
```

```
            shape = new Ellipse2D.Double(point.getLocation().x,
point.getLocation().y, DIAMETER, DIAMETER);
            tokens = new ArrayList<Token>();
            outputArcs = new ArrayList<Arc>();
            inputArcs = new ArrayList<Arc>();
        }

        @Override
        public void draw(Graphics g) {
            if (g != null) {
                Graphics2D graphics = (Graphics2D) g;
                g.setColor(Color.BLACK);
                graphics.draw(shape);
                for (Token token : tokens) {
                    token.draw(graphics);
                }
                for (Arc arc : outputArcs) {
                    arc.draw(g);
                }
                for (Arc arc : inputArcs) {
                    arc.draw(g);
                }
            }
        }

        @Override
        public Ellipse2D getShape() {
            return shape;
        }

        @Override
        public void setLocation(Point point) {
            shape.setFrame(point.getX(), point.getY(), DIAMETER, DIAMETER);
            for (Token token : tokens) {
                token.setLocation(point);
            }
            for (Arc arc : outputArcs) {
                arc.setPlace(this);
            }
            for (Arc arc : inputArcs) {
                arc.setPlace(this);
            }
        }

        public void addToken(Token token) {
            token.calcOffset(this);
            tokens.add(token);
        }

        public List<Token> getTokens() {
            return tokens;
        }

        public void addArc(Arc arc) {
            if (arc.isFromPlace()) {
                outputArcs.add(arc);
            } else {
                inputArcs.add(arc);
            }
        }

        public List<Arc> getOutputArcs() {
```

```
            return outputArcs;
        }

        public List<Arc> getInputArcs() {
            return inputArcs;
        }
    }
```

Лістинг Token

```
    package com.petri.shape;

    import java.awt.*;
    import java.awt.geom.Ellipse2D;

    /**
     * @author soboliev
     */
    public class Token implements ShapeI {

        private static final int DIAMETER = 15;
        private Ellipse2D shape;
        private int xOffset;
        private int yOffset;

        public Token(Point point) {
            shape = new Ellipse2D.Double(point.getLocation().x,
point.getLocation().y, DIAMETER, DIAMETER);
        }

        @Override
        public void draw(Graphics graphics) {
            if (graphics != null) {
                ((Graphics2D) graphics).draw(shape);
            }
        }

        @Override
        public Ellipse2D getShape() {
            return shape;
        }

        @Override
        public void setLocation(Point point) {
            shape.setFrame(point.getX() - xOffset, point.getY() - yOffset,
DIAMETER, DIAMETER);
        }

        public void calcOffset(Place place) {
            xOffset = (int) (place.getShape().getX() - shape.getX());
            yOffset = (int) (place.getShape().getY() - shape.getY());
        }

        public int getxOffset() {
            return xOffset;
        }

        public int getyOffset() {
            return yOffset;
        }

        public void setxOffset(int xOffset) {
```

```
            this.xOffset = xOffset;
        }

        public void setyOffset(int yOffset) {
            this.yOffset = yOffset;
        }

        public Token copy() {
            Token token = new Token(new Point((int) shape.getX(), (int)
shape.getY()));
            token.setxOffset(xOffset);
            token.setyOffset(yOffset);
            return token;
        }
    }
```

Лістинг Transition

```
    package com.petri.shape;

    import java.awt.*;
    import java.util.ArrayList;
    import java.util.List;

    /**
     * @author soboliev
     */
    public class Transition implements ShapeI {

        public static final int WIDTH = 20;
        public static final int HEIGHT = 70;
        private Rectangle shape;
        private List<Arc> outputArcs;
        private List<Arc> inputArcs;
        private boolean preferred;

        public Transition(Point point) {
            shape = new Rectangle(point.getLocation().x,
point.getLocation().y, WIDTH, HEIGHT);
            outputArcs = new ArrayList<Arc>();
            inputArcs = new ArrayList<Arc>();
        }

        @Override
        public void draw(Graphics graphics) {
            if (graphics != null) {
                graphics.setColor(Color.BLACK);
                ((Graphics2D) graphics).draw(shape);
            }
        }

        @Override
        public Rectangle getShape() {
            return shape;
        }

        public void addArc(Arc arc) {
            if (arc.isFromPlace()) {
                inputArcs.add(arc);
            } else {
                outputArcs.add(arc);
            }
```

```
    }

    public List<Arc> getOutputArcs() {
        return outputArcs;
    }

    public List<Arc> getInputArcs() {
        return inputArcs;
    }

    @Override
    public void setLocation(Point point) {
        shape.setLocation(point);
        for (Arc arc : outputArcs) {
            arc.setTransition(this);
        }
        for (Arc arc : inputArcs) {
            arc.setTransition(this);
        }
    }

    public boolean isPreferred() {
        return preferred;
    }

    public void setPreferred(boolean preferred) {
        this.preferred = preferred;
    }
```

Лістинг PlaceController

```
package com.petri.controller;

import com.petri.shape.*;
import java.awt.Graphics;
import java.awt.Point;
import java.util.ArrayList;
import java.util.List;

/**
 * @author soboliev
 */
public class PlaceController {

    private List<Place> places;

    public PlaceController() {
        places = new ArrayList<Place>();
    }

    public void addPlace(Place place) {
        places.add(place);
    }

    public void drawPlaces(Graphics graphics) {
        for (int i = 0; i < places.size(); i++) {
            places.get(i).draw(graphics);
            if (graphics != null) {
                graphics.drawString("P" + i, (int)
places.get(i).getShape().getX(), (int) places.get(i).getShape().getY());
            }
        }
    }
```

```
        public void addToken(Point point) {
            Token token = new Token(point);
            for (Place place : places) {
                if (place.getShape().contains(token.getShape().getBounds()))
{
                    place.addToken(token);
                    return;
                }
            }
        }

        public Place getObjectFromPoint(Point point) {
            for (Place place : places) {
                if (place.getShape().contains(point)) {
                    return place;
                }
            }
            return null;
        }

        List<Place> getPlaces() {
            return places;
        }
    }
```

Лістинг TransitionController

```
    package com.petri.controller;

    import com.petri.shape.Transition;
    import java.awt.Graphics;
    import java.awt.Point;
    import java.util.ArrayList;
    import java.util.List;

    /**
     * @author soboliev
     */
    public class TransitionController {

        private List<Transition> transitions;

        public TransitionController() {
            transitions = new ArrayList<Transition>();
        }

        public void addTransition(Transition transition) {
            transitions.add(transition);
        }

        public void drawTransitions(Graphics graphics) {
            for (int i = 0; i < transitions.size(); i++) {
                transitions.get(i).draw(graphics);
                if (graphics != null) {
                    graphics.drawString("T" + i, (int)
transitions.get(i).getShape().getX(), (int)
transitions.get(i).getShape().getY());
```

```
                }
            }
        }

        public Transition getObjectFromPoint(Point point) {
            for (Transition transition : transitions) {
                if (transition.getShape().contains(point)) {
                    return transition;
                }
            }
            return null;
        }

        List<Transition> getTransitions() {
            return transitions;
        }
    }
```

Лістинг MatrixController

```
    package com.petri.controller;

    import com.petri.shape.*;
    import com.petri.view.MatrixView;
    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
    import java.util.List;
    import javax.swing.JOptionPane;

    /**
     * @author soboliev
     */
    public class MatrixController {

        private MatrixView view;
        private int[][] c;
        private int[][] f;
        private int[][] h;
        private int[] initMarking;

        public void calculateMatrix(List<Place> places, List<Transition>
transitions) {
            f = calcF(places, transitions);
            h = calcH(places, transitions);
            c = subtractionOfMatrices(h, f);
            System.out.println("Matrix c");
            printArray(c);
            initMarking = calcInitMarking(places);
        }

        private int[] calcInitMarking(List<Place> places) {
            int[] result = new int[places.size()];
            System.out.println("Init marking matrix :");
            for (int i = 0; i < places.size(); i++) {
                if (!places.get(i).getTokens().isEmpty()) {
                    result[i] = 1;
                }
                System.out.print(result[i] + " ");
            }
            System.out.println("");
            return result;
        }
```

```
        private int[][] calcF(List<Place> places, List<Transition>
transitions) {
            int[][] result = new int[places.size()][transitions.size()];
            for (Place place : places) {
                for (Arc arc : place.getOutputArcs()) {
                    for (Transition transition : transitions) {
                        if (arc.getTransition().equals(transition)) {

result[places.indexOf(place)][transitions.indexOf(transition)] = 1;
                        }
                    }
                }
            }


            System.out.println("Matrix F :");
            printArray(result);
            result = transposeMatrix(result);
            System.out.println("Matrix Ft");
            printArray(result);
            return result;
        }

        private int[][] calcH(List<Place> places, List<Transition>
transitions) {
            int[][] result = new int[transitions.size()][places.size()];
            for (Transition transition : transitions) {
                for (Arc arc : transition.getOutputArcs()) {
                    for (Place place : places) {
                        if (arc.getPlace().equals(place)) {

result[transitions.indexOf(transition)][places.indexOf(place)] = 1;
                        }
                    }
                }
            }
            System.out.println("Matrix H : ");
            printArray(result);
            return result;
        }

        private int[][] subtractionOfMatrices(int[][] a, int[][] b) {
            int m = a.length;
            int n = a[m - 1].length;
            int[][] result = new int[m][n];
            for (int i = 0; i < a.length; i++) {
                for (int j = 0; j < a[i].length; j++) {
                    result[i][j] = a[i][j] - b[i][j];
                }
            }
            return result;
        }

        private int[][] transposeMatrix(int[][] input) {
            int m = input.length;
            int n = input[m - 1].length;
            int[][] result = new int[n][m];

            for (int i = 0; i < input.length; i++) {
                for (int j = 0; j < input[i].length; j++) {
                    result[j][i] = input[i][j];
```

```
                }
            }
            return result;
        }

        private void printArray(int[][] matrix) {
            for (int i = 0; i < matrix.length; i++) {
                for (int j = 0; j < matrix[i].length; j++) {
                    System.out.print(matrix[i][j] + " ");
                }
                System.out.println("");
            }
        }

        public void show() {
            initView();
            view.setVisible(true);
        }

        private void initView() {
            view = new MatrixView();
            initConfirmButton();
            initCancelButton();
        }

        private void initConfirmButton() {
            view.getConfirmButton().addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    if (view.getTextField().getText().isEmpty()) {
                        JOptionPane.showMessageDialog(view, "Matrix not
entered");
                    } else {
                        String text = view.getTextField().getText();
                        String[] split = text.split(" ");
                        int[] vector = new int[split.length];
                        for (int i = 0; i < split.length; i++) {
                            vector[i] = Integer.valueOf(split[i]);
                        }
                        System.out.println("Entered vector - ");
                        for (String string : split) {
                            System.out.print(string + " ");
                        }
                        System.out.println("");

                        if (vector.length == c.length) {
                            int[] multiplyVectorToMatrix =
multiplyVectorToMatrix(vector, c);

                            if (multiplyVectorToMatrix.length ==
initMarking.length) {
                                int[] result = summationVectors(initMarking,
multiplyVectorToMatrix);

                                String resultString = "Result vector :";
                                for (int i : result) {
                                    resultString += Integer.toString(i) + "
";
                                }
                                JOptionPane.showMessageDialog(view,
resultString);
```

```
                            }
                        } else {
                            JOptionPane.showMessageDialog(view, "Vector size
is incompatible");
                        }
                        closeView();
                    }

                }
            });
        }

        private int[] summationVectors(int[] first, int[] second) {
            int[] result = new int[first.length];
            for (int i = 0; i < first.length; i++) {
                result[i] = first[i] + second[i];
            }
            return result;
        }

        private int[] multiplyVectorToMatrix(int[] vector, int[][] matrix) {
            int m = matrix.length;
            int n = matrix[m - 1].length;
            int[] result = new int[n];
            for (int i = 0; i < n; i++) {
                int temp = 0;
                for (int j = 0; j < matrix.length; j++) {
                    temp += matrix[j][i] * vector[j];
                }
                result[i] = temp;
            }

            System.out.println("Multiply ^");
            for (int i : result) {
                System.out.print(i + " ");
            }
            return result;
        }

        private void initCancelButton() {
            view.getCancelButton().addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    closeView();
                }
            });
        }

        private void closeView() {
            view.setVisible(false);
            view.dispose();
        }
    }
```

Лістинг PetriController

```
package com.petri.controller;

import com.petri.shape.*;
import com.petri.view.PetriView;
```

```
import com.petri.view.TransitionPreferPopup;
import java.awt.Point;
import java.awt.event.*;
import java.util.List;
import javax.swing.SwingUtilities;

/**
 * @author soboliev
 */
public class PetriController {

    private PetriView view;
    private State state;
    private PlaceController placeController;
    private TransitionController transitionController;
    private ShapeI shapeToDrag;
    private Arc arcToDraw;

    public PetriController() {
        initSubControllers();
        initView();
        initDrawer();
    }

    private void initSubControllers() {
        placeController = new PlaceController();
        transitionController = new TransitionController();
    }

    private void initView() {
        view = new PetriView();
        initNewButton();
        initAddPlaceButton();
        initDeletePlaceButton();
        initAddTransitionButton();
        initAddTokenButton();
        initAddArcButton();
        initDragButton();
        initRunButton();
        initMatrixMenu();
    }

    private void initNewButton() {
        view.getToolBar().getNewButton().addActionListener(new
ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                placeController.getPlaces().clear();
                transitionController.getTransitions().clear();
                arcToDraw = null;
                shapeToDrag = null;
                redrawShapes();
            }
        });
    }

    private void initAddPlaceButton() {
        view.getToolBar().getAddPlaceButton().addActionListener(new
ActionListener() {

            @Override
```

```
                public void actionPerformed(ActionEvent e) {
                    setState(State.ADD_PLACE);
                }
            });
        }

        private void initDeletePlaceButton() {
            view.getToolBar().getDeletePlaceButton().addActionListener(new
ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    setState(State.DELETE_PLACE);
                }
            });
        }

        private void initAddTransitionButton() {
            view.getToolBar().getAddTransitionButton().addActionListener(new
ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    setState(State.ADD_TRANSITION);
                }
            });
        }

        private void initAddTokenButton() {
            view.getToolBar().getAddTokenButton().addActionListener(new
ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    setState(State.ADD_TOKEN);
                }
            });
        }

        private void initAddArcButton() {
            view.getToolBar().getAddArcButton().addActionListener(new
ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    setState(State.ADD_ARC);
                }
            });
        }

        private void initDragButton() {
            view.getToolBar().getDragButton().addActionListener(new
ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    setState(State.DRAG);
                }
            });
        }

        private void initRunButton() {
```

```
            view.getToolBar().getRunButton().addActionListener(new
ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    setState(State.RUN);
                }
            });
        }

        private void initMatrixMenu(){
            view.getMatrixMenu().addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    MatrixController controller = new MatrixController();
                    controller.calculateMatrix(placeController.getPlaces(),
transitionController.getTransitions());
                    controller.show();
                }
            });
        }

        private void initDrawer() {
            view.getDrawer().addMouseListener(new MouseAdapter() {

                @Override
                public void mousePressed(MouseEvent e) {
                    if (state == State.DRAG) {
                        chooseShapeToDraw(e.getPoint());
                    } else if (state == State.ADD_ARC) {
                        Place place =
placeController.getObjectFromPoint(e.getPoint());
                        if (place != null) {
                            arcToDraw = new Arc(place);
                            redrawShapes();
                            return;
                        }

                        Transition transition =
transitionController.getObjectFromPoint(e.getPoint());
                        if (transition != null) {
                            arcToDraw = new Arc(transition);
                            redrawShapes();
                        }
                    }
                }

                @Override
                public void mouseReleased(MouseEvent e) {
                    Transition transition =
transitionController.getObjectFromPoint(e.getPoint());
                    if (transition != null && arcToDraw != null &&
arcToDraw.isFromPlace()) {
                        arcToDraw.setTransition(transition);
                        arcToDraw.addToPlaceAndTrnasition();
                        arcToDraw = null;
                        redrawShapes();
                        return;
                    }
```

```
                    Place place =
placeController.getObjectFromPoint(e.getPoint());
                    if (place != null && arcToDraw != null &&
!arcToDraw.isFromPlace()) {
                        arcToDraw.setPlace(place);
                        arcToDraw.addToPlaceAndTrnasition();
                        arcToDraw = null;
                        redrawShapes();
                        return;
                    }
                    arcToDraw = null;
                    redrawShapes();
                }

                @Override
                public void mouseClicked(MouseEvent e) {
                    if (SwingUtilities.isRightMouseButton(e)) {
                        Transition transition =
transitionController.getObjectFromPoint(e.getPoint());
                        if (transition != null) {
                            TransitionPreferPopup popup = new
TransitionPreferPopup(e.getLocationOnScreen());
                            popup.setTransition(transition);
                            popup.setVisible(true);
                        }
                        return;
                    }
                    if (state == State.ADD_PLACE) {
                        placeController.addPlace(new Place(e.getPoint()));

placeController.drawPlaces(view.getDrawer().getGraphics());
                    } else if (state == State.ADD_TRANSITION) {
                        transitionController.addTransition(new
Transition(e.getPoint()));

transitionController.drawTransitions(view.getDrawer().getGraphics());
                    } else if (state == State.ADD_TOKEN) {
                        placeController.addToken(e.getPoint());

placeController.drawPlaces(view.getDrawer().getGraphics());
                    } else if (state == State.DELETE_PLACE) {
                        Place place =
placeController.getObjectFromPoint(e.getPoint());
                        if (place != null) {
                            placeController.getPlaces().remove(place);
                            redrawShapes();
                        }
                    } else if (state == State.RUN) {
                        replaceToken(e.getPoint());
                    }
                }
            });

            view.getDrawer().addMouseMotionListener(new MouseAdapter() {

                @Override
                public void mouseDragged(MouseEvent e) {
                    if (state == State.DRAG && shapeToDrag != null) {
                        shapeToDrag.setLocation(e.getPoint());
                    } else if (state == State.ADD_ARC && arcToDraw != null)
{
                        arcToDraw.setLocation(e.getPoint());
```

```
                    }
                    redrawShapes();
                }
            });
        }

        private void replaceToken(Point point) {
            Place startPlace = placeController.getObjectFromPoint(point);
            if (startPlace != null && !startPlace.getTokens().isEmpty()) {
                if (!startPlace.getOutputArcs().isEmpty()) {
                    Transition transition =
getPreferredTransition(startPlace.getOutputArcs());
                    if (!transition.getOutputArcs().isEmpty()) {
                        Token token =
startPlace.getTokens().get(startPlace.getTokens().size() - 1);
                        startPlace.getTokens().remove(token);
                        for (Arc outputArc : transition.getOutputArcs()) {
                            Place finishPlace = outputArc.getPlace();
                            Token copyToken = token.copy();
                            copyToken.setLocation(new Point((int)
finishPlace.getShape().getX(), (int) finishPlace.getShape().getY()));
                            finishPlace.addToken(copyToken);
                        }
                        redrawShapes();
                    }
                }
            }
        }

        private Transition getPreferredTransition(List<Arc> arcs){
            Transition result= arcs.get(0).getTransition();
            for (Arc arc : arcs) {
                if(arc.getTransition().isPreferred()){
                    result = arc.getTransition();
                    break;
                }
            }
            return result;
        }

        private void chooseShapeToDraw(Point point) {
            ShapeI                    objectFromPoint                    =
placeController.getObjectFromPoint(point);
            if (objectFromPoint != null) {
                shapeToDrag = objectFromPoint;
                return;
            }
            objectFromPoint = transitionController.getObjectFromPoint(point);
            if (objectFromPoint != null) {
                shapeToDrag = objectFromPoint;
            }
        }

        private void redrawShapes() {
            view.getDrawer().clearGraphics();
            placeController.drawPlaces(view.getDrawer().getGraphics());

transitionController.drawTransitions(view.getDrawer().getGraphics());
            if (arcToDraw != null) {
                arcToDraw.draw(view.getDrawer().getGraphics());
            }
        }
```

```
private void setState(State state) {
    this.state = state;
    System.out.println("State changed to " + this.state);
}

PetriView getView() {
    return view;
}

State getState() {
    return state;
}

PlaceController getPlaceController() {
    return placeController;
}

TransitionController getTransitionController() {
    return transitionController;
}

ShapeI getShapeToDrag() {
    return shapeToDrag;
}

Arc getArcToDraw() {
    return arcToDraw;
}

public static void main(String[] args) {
    new PetriController().getView().setVisible(true);
}
```